PROYECTO FIN DE MÁSTER

POSGRADO EN INFORMÁTICA

UNIVERSIDAD DE ALMERÍA

ESCUELA SUPERIOR DE INGENIERÍA

MÁSTER EN INFORMÁTICA INDUSTRIAL

IMPLEMENTACIÓN EFICIENTE DEL TEOREMA CHINO DEL RESTO

Curso 2011/2012

Alumno/a:
Jose Manuel Arrufat González

Director/es:
Dr. J.A. Álvarez Bermejo
Dr. J.A. López Ramos

UNIVERSIDAD DE ALMERÍA
ESCUELA SUPERIOR DE INGENIERÍA
Departamento de Arquitectura de Computadores y Electrónica

TRABAJO FIN DE MÁSTER
MÁSTER EN INFORMÁTICA INDUSTRIAL
POSGRADO EN INFORMÁTICA

IMPLEMENTACIÓN EFICIENTE DEL TEOREMA CHINO DEL RESTO

Jose Manuel Arrufat González

Dirigida por: Dr. José Antonio Álvarez Bermejo y Dr. Juan Antonio López Ramos

Almería, Septiembre 2012

TRABAJO FIN DE MÁSTER
MÁSTER EN INFORMÁTICA INDUSTRIAL
POSGRADO EN INFORMÁTICA

IMPLEMENTACIÓN EFICIENTE DEL TEOREMA CHINO DEL RESTO

por
Jose Manuel Arrufat González

Para la obtención del
Título del Máster en Informática Industrial
Posgrado en Informática

Director	**Director**	**Autor**
Dr. José Antonio Álvarez Bermejo	Dr. Juan Antonio López Ramos	Jose Manuel Arrufat González

Agradecimientos

Me voy a permitir el lujo de usar esta sección para ponerme sentimental, y agradecer a todas aquellas personas que me han acompañado a lo largo de este gran viaje, que no hace más que cambiar de vía, desde la de la educación hacia la laboral, y dedicarles unas breves líneas. Porque se lo merecen.

Me gustaría comenzar agradeciendo de corazón todo su apoyo, confianza, aguante y cariño durante todo este tiempo a mis padres y hermana, porque sin ellos no estaría hoy donde estoy, ni sería la persona que he llegado a ser. Gracias.

A mi familia, que no es poca, por creer y confiar siempre en mí, dándome su apoyo desde el principio y animándome a continuar y avanzar siempre, incluso en los momentos en los que nadie, ni si quiera yo, confiaba en que las cosas pudieran salir bien.

A mi pareja, por entender mi situación y comprenderme, y por estar siempre ahí para apoyarme y levantarme cuando era necesario.

A compañeros y amigos, tanto de dentro como de fuera de la universidad, por aguantar nervios, largas temporadas de incomunicación, y charlas monotemáticas realmente aburridas. Mirando atrás, me doy cuenta de lo mucho que han hecho por mí, y de lo grandes que son. No puedo olvidarme de Israel de la Plata, gran amigo y compañero, con el que descubrí que había otra forma de hacer las cosas, y que, trabajando, podemos llegar lejos tan lejos como queramos, él ya lo ha conseguido. Junto a él, tuve también la suerte de compartir concursos y eventos con José Aguado, el cual se ha convertido con el tiempo en mi compañero de trabajo, y que ha tenido que soportar muchas de mis peculiaridades. Gracias por poner siempre la nota de razón y dedicar tu tiempo a escucharme y apoyarme. Además, querría acordarme de Manuel Godoy, compañero y amigo desde que comenzamos juntos la aventura de la Ingeniería Informática en 4º, gracias al cual he aprendido mucho. Gracias por confiar en mí.

A todos aquellos profesores que me han acompañado en este viaje, desde 1º hasta 5º, para poder llegar a este master, y cumplir mi objetivo. En especial a los que me apoyaron y confiaron en mí, como mi director de PFC, Francisco de Asís Guindos Rojas, por darme a conocer nuevas tecnologías, que me han llevado a este TFM, y aconsejarme siempre buscando lo mejor para mí.

A José Antonio Álvarez Bermejo, por confiar en mí.

A Juan Antonio López Ramos, por su gran apoyo y tutela en este TFM, sin el cual ahora no estaría escribiendo estas líneas.

A Fernando Reche Lorite, mi actual y primer jefe, porque gracias a su confianza en dos estudiantes con ganas de aprender y de emprender nuevos retos, he tenido la posibilidad de lanzarme a la aventura que ha supuesto este master para mí.

A todos, gracias.

ÍNDICE

Implementación Eficiente del Teorema Chino del Resto

Jose Manuel Arrufat González, *Master en Informática Industrial, Escuela Politécnica Superior, Universidad de Almería*

***Resumen*—Estudio, diseño e implementación de nuevos enfoques paralelos software del Teorema Chino del Resto sobre diferentes plataformas y arquitecturas de tipo paralelo. Algunos de estos diseños típicos generan problemas conocidos de almacenamiento y gestión de la memoria en su ejecución al manejar Enteros Grandes. Este problema lleva al diseño de nuevos métodos más eficientes y refinados que vienen a solventar estas cuestiones de forma original y escalable. Los resultados obtenidos no solo mejoran de forma sustancial la implementación secuencial del mismo, sino que representan un avance en eficiencia, rendimiento y escalabilidad respecto a otras alternativas existentes actualmente.**

***Abstract*— This work presents a study, design and implementation of a new parallel software approach to the Chinese Remainder Theorem on different parallel architectures. Some of the known and legacy implementation described suffer from storage and memory management issues when handling big integers, affecting substantially to performance. This problem leads to the design of a new and more efficient and refined method to address these issues in a scalable way. The results not only substantially overcome the sequential implementation but represent an doubtlessly enhancement in efficiency, performance and scalability regarding existing alternatives.**

***Palabras Clave*—Arquitecturas paralelas, criptografía, programación paralela, Teorema Chino del Resto**

***Index Terms*—Chinese Remainder Theorem, cryptography, parallel architectures, parallel programming.**

I. Introducción

El modelado de problemas que se pretenden resolver con métodos computacionales no sólo se enfrenta al reto de reflejar con coherencia el problema para que pueda ser computacionalmente tratado, sino que además ha de ser representado de manera correcta y tratable. A esto se le deben sumar las estructuras de datos que sustentan el modelo computacional.

En cualquier caso, a la complejidad que caracteriza al modelado de un fenómeno propio de la ciencia o del ámbito de la ingeniería hay que sumar la correcta elección en la forma de representarlo. Modelar algebraicamente un problema es una solución para llevar al campo computacional un problema concreto. La naturaleza de ciertos métodos algebraicos usados comúnmente como es el caso del Teorema del Resto Chino o Chinese Remainder Theorem (CRT) ([LK1], [RK] y [W]), hace necesaria la aplicación de técnicas que aceleren el proceso, pues su naturaleza secuencial impide una aplicación eficiente del modelo que se desea digitalizar, algunos ejemplos puede encontrarse en [TSA], [LCh], [O] [ChKL] y [TSA].

Existen muchas aproximaciones al problema de acelerar el CRT, las más interesantes son las que no precisan de hardware especialmente dedicado que acelere las operaciones. Hoy día las arquitecturas paralelas suponen un activo importante en la aplicación de estas técnicas algebraicas aplicadas a problemas que se pretenden resolver computacionalmente.

Un problema abierto, dado el uso extensivo de intensivo que se está haciendo de las comunicaciones a través de la Red, es el de la protección de la información que se remite a través de canales donde cualquiera puede tener acceso, lo cual se requiere en numerosas aplicaciones de un modo eficiente dado el auge de la distribución de contenidos en streaming. Los denominados esquemas multicas seguros (ver [ChCh], [RH] o [ZJ]) son la solución preferida actualmente. Dichos métodos permiten que un usuario envíe contenidos a una pluralidad de usuarios de forma segura y eficiente y todos los usuarios recuperan la información original utilizando una clave de sesión común que se renueva cada vez que un usuario se une o abandona el grupo que se encuentra compartiendo dicha información. Este tipo de métodos se encuentra muy extendido en aplicaciones como la IPTV [LZJ]. El esquema de multicas seguro introducido en [ChiCh] demuestra una simplicidad y unos parámetros en cuanto a costes de la comunicación superiores a la mayoría de los existentes y actualmente utilizados. Sin embargo, los requerimientos computacionales en el servidor de claves hacen que el sistema se vuelva ineficiente cuando el número de usuarios es grande, [KM], como es el caso de las principales aplicaciones de este tipo de esquemas de comunicaciones.

Este problema se debe al uso del CRT para la generación de mensajes de refresco. Notemos que l método anteriormente citado en [LZJ] se basa en un polinomio interpolador sobre un cuerpo finito, lo cual puede considerarse como un caso particular del CRT. En dicho caso, la ineficacia de éste es suplida por una distribución de los usuarios por grupos en forma de árbol. Otros criptosistemas que hacen uso del CRT son, por ejemplo, los introducidos en [LK2] y [LXWT].

A. *Teorema Chino de los Restos*

En cuanto al Teorema Chino de los Restos, podemos definirlo brevemente según [RK] *como:*

Teorema Chino de los Restos

Dados $m_1, m_2, \dots, m_n$ enteros positivos mayores que uno y primos relativos dos a dos, mayores que $a_1, a_2, \dots, a_n$enteros cualesquiera. Entonces el sistema

$$x \equiv a_1 (mod\ m_1)$$
$$x \equiv a_2 (mod\ m_2)$$
$$\vdots$$
$$x \equiv a_n (mod\ m_n)$$

Tiene una única solución modulo $m = m_1 m_2 \dots m_n$.

Existe una fórmula bastante eficiente para el cálculo de una solución de un sistema como el que acabamos de citar para el caso de números pequeños tal y como se pone de manifiesto en el siguiente ejemplo. Sin embargo, como veremos en los siguientes capítulos el problema se complica al considerar números del orden de los que se requieren en aplicaciones computacionales en la actualidad, así como un número grande de congruencias que componen el sistema tal y como puede suceder en el caso de aplicaciones multicast, también citadas anteriormente.

Veamos pues un ejemplo que nos permita vislumbrar cómo obtener una solución, así como la problemática en el tratamiento de la información cuando este método se aplica a situaciones reales en el ámbito computacional y de las comunicaciones.

Ejemplo CRT

Dado el siguiente sistema de ecuaciones de congruencias lineales

$$x \equiv 2 (mod\ 3)$$
$$x \equiv 3 (mod\ 5)$$
$$x \equiv 2 (mod\ 7)$$

La resolución del sistema por medio del Teorema del Resto Chino resulta como sigue

$$m = 3 * 5 * 7 = 105$$
$$M_1 = \frac{m}{3} = 35, M_2 = \frac{m}{5} = 21, M_3 = \frac{m}{7} = 15$$

Se realiza el cálculo de los inversos correspondientes a los nuevos módulos

$$y_1 = 2\ es\ inverso\ de\ M_1 (mod\ 3)$$
$$y_2 = 1\ es\ inverso\ de\ M_2 (mod\ 5)$$
$$y_3 = 1\ es\ inverso\ de\ M_3 (mod\ 7)$$

Soluciones son aquellos tales que

$$X = a_1 M_1 y_1 + a_2 M_2 y_2 + a_3 M_3 y_3$$
$$X = 2 * 35 * 2 + 3 * 21 * 1 + 2 * 15 * 1 = 233$$
$$233 = 23 (mod\ 105)$$

Luego 23 es el número entero positivo más pequeño cuyo residuo es dos cuando se divide por tres, tiene resto tres cuando se divide por cinco y tiene resto dos cuando se divide por siete.

B. *Teorema Chino de los Restos Generalizado*

Además del CRT, se presente en alguna de las fuentes implementaciones eficientes sobre distintas arquitecturas del Teorema Chino de los Restos Generalizado. Este presenta una variación sobre el anterior que se podrá vislumbrar a continuación.

Teorema Chino de los Restos Generalizado

Sean $m_1, m_2, \dots, m_k$ enteros positivos mayores que uno y sean $a_1, a_2, \dots, a_k$enteros cualesquiera. Entonces el sistema

$$x \equiv a_1 (mod\ m_1)$$
$$x \equiv a_2 (mod\ m_2)$$
$$\vdots$$
$$x \equiv a_k (mod\ m_k)$$

Tiene solución si, y sólo si, $mcd(m_i, m_j)$ divide a $a_i - a_j$ para cualesquiera $i \neq j$.

Cuando se verifica esta condición, la solución general constituye una única clase de congruencias de módulo n, donde n es el mínimo común múltiplo de los enteros $m_1, m_2, \dots, m_k$.

A partir del ejemplo anterior podemos hacer las siguientes observaciones. En primer lugar, el tamaño del problema depende del número de usuarios, k, el cual puede ser enorme, como nos daría una aplicación del CRT para la distribución de una señal cifrada de información a una pluralidad de usuarios como en el caso de la denominada IPTV ([LZJ]). De este modo, podemos comprender el interés en obtener una optimización (preferiblemente en software) del CRT.

El trabajo está estructurado como sigue: en la sección siguiente se mostrará el enfoque teórico del CRT y el problema de las paralelizaciones existentes hasta ahora, además de plantear las nuevas aproximaciones propuestas. Estas nuevas aproximaciones vienen presentadas bajo dos enfoques distintos, una desde la paralelización basada en el ejemplo del CRT, mientras que la otra solución aportada se basa en una implementación eficiente basada en modelos de diseño divide y vencerás. A continuación, se detalla el proceso de implementación y los ajustes y conversiones necesarias para llevar a buen término el desarrollo de los nuevos diseños. Estos se basan en las necesidades de las distintas plataformas de codificación elegidas, como procesadores gráficos, que requieren de medidas especiales en su forma. Seguidamente, mostramos los resultados obtenidos en una serie de pruebas reales. Estas arrojan datos significativos de mejoría frente a implementaciones secuenciales y disminuyen los tiempos de procesamiento al rango de milisegundos para grandes cantidades de datos de entrada. En el último capítulo damos cuenta de las líneas de trabajo abiertas y la orientación de los

futuros esfuerzos a llevar a cabo sobre la cuestión, mostrando algunas aplicaciones reales de nuestro diseño al caso de los esquemas multicast concerniendo la confidencialidad en la distribución de la información, así como la integridad, autentificación y no repudio de la misma. Todo este contenido ha sido presentado a las VIII Jornadas de Matemática Discreta y Algorítmica en Almería en Julio de 12 ([AAL]).

II. EL TEOREMA CHINO DE LOS RESTOS

Siguiendo [R], la resolución de congruencias lineales, de la forma $ax \equiv b\ (mod\ m)$, es una tarea esencial en el estudio de la teoría de números y sus aplicaciones, tal como podemos observar en [AL], [ZJ], **[**O**]** y **[**ChKL**],** y, de forma distinta pero bajo el mismo principio, en **[**TSA**]**, así como la resolución de ecuaciones lineales juega un papel importante en el cálculo y el álgebra lineal. Para resolver congruencias lineales, empleamos inversas modulo m. En nuestro caso vamos a utilizar el conocido como algoritmo de Euclides para encontrar estos inversos. Una vez obtenida la inversa, podemos resolver la congruencia multiplicando ambos lados de la misma por el inverso obtenido, tal y como se hace con una ecuación planteada sobre los números racionales.

Los sistemas simultáneos de congruencias lineales han sido estudiados desde tiempos antiguos. Existen documentos que nos muestran como el matemático chino Sun-Tsu los estudió ya allá por el siglo I. Vamos a presentar pues una forma de resolver estos sistemas conocida (en su honor) como Teorema Chino de los Restos, así como su demostración, la cual nos inspirará para nuestras nuevas propuestas presentadas en esta memoria.

A. Congruencias Lineales

Una congruencia es una relación de equivalencia dentro de un conjunto, es decir, una relación binaria que satisface las propiedades reflexiva, simétrica y transitiva [RK]. Cuando el conjunto considerado es el de los números enteros, el conjunto determinado por las clases de equivalencia se conoce como Z_m y solemos escribir $a \equiv b\ (mod\ m)$ para indicar la relación de a y b mediante el entero m y que viene determinada en este caso por que "m divide a la diferencia $a - b$". Dicho conjunto Z_m viene determinado por las clases de restos módulo m, es decir, los elementos de Z_m son $0, \dots, m-1$.

Lo que a lo largo de esta memoria referiremos por congruencia lineal o simplemente por congruencia es, de hecho una ecuación lineal en congruencias de la forma $ax \equiv b\ (mod\ m)$, donde m es un entero positivo, a y b son enteros, y x es una variable. Resolver una congruencia nos equivale a encontrar un elemento dentro del conjunto finito Z_m.

Para resolver estas congruencias, una posible solución es usar un entero $\bar{a}$ de tal forma que $\bar{a}a \equiv 1(mod\ m)$, si este entero existe. Este nuevo entero es lo que denominamos la inversa de a modulo m. El Teorema 1 garantiza que esta inversa existe cuando a y m son primos relativos.

Teorema 1

Si a y m son primos relativos y $m > 1$, entonces el inverso de a modulo m existe. Adicionalmente, dicho inverso es único módulo m.

Demostración

En base al Teorema de Bézout, dado que el $gcd(a, m) = 1$, existen los enteros s y t tal que

$$sa + tm = 1$$

Esto implica que

$$sa + tm \equiv 1(mod\ m)$$

Dado que $tm \equiv 0(mod\ m)$, tenemos que

$$sa \equiv 1(mod\ m)$$

Luego s es consecuentemente un inverso de a módulo m. La unicidad del mismo módulo m es consecuencia de la unicidad del inverso en cualquier grupo, como es el caso de Z_m.

B. Teorema Chino de los Restos

Los sistemas de congruencias lineales aparecen en múltiples contextos, tales como aritmética de enteros largos o problemas criptográficos, como podemos ver en [WWX] y [MY]. Debido a ello es necesario un método de resolución sencillo y abordable.

El Teorema Chino de los Restos, en adelante CRT [RK], denominado así tras la herencia china de problemas entorno a sistemas de congruencias lineales, enuncia que cuando los módulos de un sistema de congruencias lineales son primos relativos dos a dos, existe una única solución del sistema modulo el producto de los módulos.

Teorema Chino de los Restos

Dados los enteros $m_1, m_2, \dots, m_n$ enteros positivos mayores que uno y primos relativos dos a dos y cualesquiera enteros $a_1, a_2, \dots, a_n$ enteros cualesquiera. Entonces el sistema

$$x \equiv a_1(mod\ m_1)$$
$$x \equiv a_2(mod\ m_2)$$
$$\vdots$$
$$x \equiv a_n(mod\ m_n)$$

Tiene una única solución modulo $m = m_1 m_2 \dots m_n$.

Demostración

Para probar este teorema necesitamos demostrar que existe una solución, y que además es única modulo m.

Para construir una solución simultánea, primero tenemos

$$M_k = m/m_k$$

Para $k = 1, 2, \dots, n$. M_k es el producto de los módulos excepto m_k. Dado que m_i y m_k no tienen factores en común mayores que 1 cuando $i \neq k$, quiere decir que

$gcd(m_k, M_k) = 1$. En consecuencia, dado el Teorema 1, sabemos que existe un entero y_k, inverso de M_k modulo m_k, tal que

$$M_k y_k \equiv 1 (mod\ m_k)$$

Para construir la solución simultánea, formamos la suma

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots + a_n M_n y_n$$

Ahora demostramos que x es una solución simultánea. Primero, veamos que, debido a que $M_j \equiv 0 (mod\ m_k)$ cuando $i \neq k$, todos los términos excepto el término k en la suma son congruentes a 0 modulo m_k. Dado que $M_k y_k \equiv 1 (mod\ m_k)$ tenemos que

$$x \equiv a_k M_k y_k \equiv a_k (mod\ m_k)$$

para $k = 1, 2, \ldots, n$. Luego x es solución simultánea de las n congruencias.

La demostración anterior del CRT nos prueba que la fórmula utilizada para la resolución del mismo es correcta. Sin embargo podemos dar otra demostración existencial de este hecho y que nos inspirará para nuestra propuesta de mejora de los resultados que aparecen en las siguientes secciones. Esta demostración es como sigue:

Teorema 2

Sean $m_1, m_2, \ldots, m_n$ números enteros mayores que 1 y primos relativos dos a dos. Entonces el sistema de congruencias $\equiv_{m_1} u_1, x \equiv_{m_2} u_2, \ldots, x \equiv_{m_n} u_n$ tiene solución única módulo $m_1 m_2 \ldots m_n$.

Demostración

Hagamos inducción sobre n. Para n=2, la solución general de la ecuación $x \equiv_m u$ es $x = u + mk$ con k un número entero. Así pues, $u + mk \equiv_{m'} v$ si y solo si $mk \equiv_{m'} v - u$. Ahora bien, por hipótesis $(m, m') = 1$, con lo que existe k_0 tal que $mk_0 \equiv_{m'} v - u$ y así el sistema tiene solución. Supongamos ahora que x_1 y x_2 son dos soluciones del sistema de congruencias. Entonces $x_1 \equiv_m x_2$ y $x_1 \equiv_{m'} x_2$, por lo que $x_1 \equiv_{mm'} x_2$.

Supongamos que el resultado es cierto para un sistema con $k - 1$ congruencias como el anterior. Sea pues un sistema con k congruencias como el enunciado. Si consideramos el sistema compuesto por las $k - 1$ primeras, por la hipótesis de inducción, dicho sistema tiene solución x_0, que es única módulo $m_1 m_2 \ldots m_{k-1}$. Para obtener una solución del sistema de k congruencias, hemos de resolver el sistema formado por las congruencias $x \equiv_{m_1 m_2 \ldots m_{k-1}} x_0$ y $x \equiv_{m_k} u_k$. Como $m_1, m_2, \ldots, m_k$ son primos relativos, se tiene que $m_1, m_2, \ldots, m_{k-1}$ y m_k también lo son. Pero entonces, el teorema del resto chino nos garantiza la existencia de una solución del sistema anterior que es única módulo $m_1 m_2 \ldots m_k$. Pero cualquier solución de este sistema, x_1, lo es también del sistema formado por las k congruencias, ya que si $x_1 \equiv_{m_1 m_2 \ldots m_{k-1}} x_0$, se tiene que $x_1 \equiv_{m_i} x_0$ para todo $i = 1, \ldots, k - 1$, con lo que $x_1 \equiv_{m_i} u_i$ para todo $i = 1, \ldots, k$ pues x_0 era solución de las $k - 1$ primeras y $x_1$es también solución de $x \equiv_{m_k} u_k$.

C. Paralelizaciones Existentes

Debido a los problemas derivados de la búsqueda de una solución en el CRT que pueden observarse en ambas aproximaciones citadas del uso del CRT como pueden ser un número grande de congruencias a resolver, así como el tamaño de los enteros utilizados, lo que produce problemas de representación y tratamiento de los parámetros necesarios en el en el primero de los dos casos, y en el segundo los derivados de un proceso iterativo con dicho número y tamaño de los datos, se hace necesario el uso de tecnologías de paralelización tanto hardware como software para solventar problemas de eficiencia.

Es en este campo donde han proliferado multitud de versiones aplicadas a distintos aspectos de la computación y la criptografía, resultando estas dos vertientes de implementación hardware y software. En la actualidad, los esfuerzos se centran en el desarrollo software, debido tanto a la rapidez de implementación, como a los tiempos derivados de su diseño y prueba. Además, las implementaciones hardware están orientadas a un cometido muy específico, y su integración en nuevos sistemas de distinta aplicación, o en el mismo sistema con cambios en su estructura o en los datos procesados, genera un problema derivado de rediseño de todo el componente hardware, así como de nuevos procesos de construcción y test de la nueva arquitectura. Bien es cierto que, no olvidando los problemas aparecidos derivados de esta metodología, el rendimiento de un componente hardware diseñado y creado específicamente para este cometido ofrece unos resultados de eficiencia mucho mayores que cualquier aplicación software que se pueda desarrollar.

Las nuevas implementaciones que han seguido la tendencia del desarrollo de software han proliferado sobre las posibilidades que las arquitecturas que han inundado el mercado en los últimos años ofrecen. En particular, los desarrollos han aprovechado la potencia de los nuevos procesadores multi-core, o multi-núcleo, que proporcionan la posibilidad de contar con un equipo provisto de capacidades de computación paralela a un bajo coste. De igual manera se han realizado esfuerzos de considerables resultados sobre otras nuevas plataformas de computación paralela, tales como las tarjetas gráficas, usando el alto índice de paralelismo implícito en los procesadores, GPU, integrados en estos dispositivos, que, además, están presentes en cualquier computador actual. Estas unidades quedan pues desaprovechadas, por lo que la investigación sobre optimización de algoritmos tradicionalmente aplicados sobre computación secuencial o paralela tradicional esta migrando, a pequeños pasos, hacia estas nuevas estructuras.

Y sobre estas arquitecturas, y teniendo en cuenta los datos y experiencias anteriores, presentamos los resultados de los esfuerzos actuales en paralelización del CRT, dando una visión global del estado de las investigaciones al respecto.

Esfuerzos Hardware

Las estrategias de resolución de problemas e implantación de algoritmos y métodos mediante componentes físicos, han sido la vía más usada y explotada históricamente, debido a la gran eficiencia desarrollada en su funcionamiento, sus bajos costes de consumo, útiles en numerosas aplicaciones, y su gran especificidad, eliminando componentes y actividades superfluas durante su funcionamiento.

Debido a ello, fue una de las principales vías de implementación del CRT, siendo aún hoy un gran campo de estudio e investigación, orientado a conseguir implementaciones eficientes, aunando los requisitos de funcionalidad, consumo y tamaño, tal y como podemos ver en [G].

De igual manera, en [TSA] podemos ver una pincelada sobre los métodos y las implementaciones hardware comúnmente usadas, así como dos nuevas alternativas basadas en mejoras sustanciales relativas al diseño de las anteriores.

Este tipo de soluciones se basan comúnmente en obtener caminos hardware en los que se reduzca considerablemente la latencia de procesamiento o de cruce de los caminos, buscando la mejora de eficiencia mediante una reorganización de las unidades presentes, o realizando un proceso de adición de nuevos componentes, que introduzcan nuevos caminos, asemejándose a la paralelización software.

Ahondando en los resultados de los diseños propuestos, podemos observar que, en simulación y para conjuntos pequeños, se obtienen resultados realmente asombrosos, gestionando las salidas en tiempos cuya medida se encuentra en el rango de los nanosegundos. Estos resultados arrojan un gran y muy positivo punto a favor de las implementaciones hardware en lo que a eficiencia se refiere, pero no debemos olvidar los costes relativos a los procesos de diseño, prueba e implementación. Además, como ya hemos comentado, su adaptabilidad es compleja y, en algunos casos, llega a ser una labor prácticamente imposible ante nuevas restricciones o necesidades.

Por tanto no es esta la vía que buscamos, ya que no es fácilmente adaptable, ni presenta una interfaz de configuración y uso fácilmente implementable en nuevas aplicaciones o procesos.

Si prestamos atención, podemos ver que ambas referencias han sido obtenidas de artículos no muy actuales. Esto es debido a que se ha producido una migración continuada, escalonada y suave hacia el campo de la implementación software, debido al avance de los últimos años en lo que a su tecnología se refiere. Por ello evaluaremos a continuación distintas propuestas de esta alternativa.

Esfuerzos Software

En contraposición a las vías basadas en construcción de componentes, encontramos aquellas que optan por el desarrollo de software sobre computadores convencionales como base para la búsqueda de la eficiencia. Distinguiremos dos vertientes bien diferenciadas. En primer lugar aquellas dedicadas al uso de plataformas CPU, que buscan aprovechar las arquitecturas multi-núcleo que han irrumpido en el mercado en los últimos tiempos. Por otro lado, consideraremos las vías cuyo banco de trabajo se componen de plataformas híbridas, donde el procesamiento GPGPU cobra gran importancia en la búsqueda de resultados.

1. Implementaciones CPU

Para dar un enfoque realista de la evolución de los resultados obtenidos sobre plataformas CPU, vamos a realizar una exposición en orden cronológico de las conclusiones obtenidas en las referencias consultadas, para presentar un proceso ascendente, en cuya cúspide enlazará con las técnicas híbridas presentadas a continuación.

1) Parallel computational algorithms for generalized Chinese remainder theorem – 2001 [LCh]

De esta manera, nuestro viaje comienza con [LCh], cuya presentación data del 2001, y donde se nos presentan algunos algoritmos de computación paralela para el Teorema Chino del Resto y el Teorema Chino del Resto Generalizado.

En concreto, se propone una arquitectura algorítmica basada en 3 procesos que realizarán las diversas tareas necesarias y sobre los que se introducirá el nivel de paralelismo deseado.

El Algoritmo 1 se usa para computar la acumulación de multiplicaciones Q y Q_i. En base a él, cada procesador arroja dos valores Q y Q_i, m siguiendo un enfoque de tipo divide y vencerás. Su complejidad computacional se encuentra en $O(log\ m)$, donde m es el número de módulos coprimos $\{q_1, q_2, \cdots, q_m\}$. Por ejemplo, un procesador realiza operaciones de multiplicación 3 veces para computar el número Q con una 8-tupla de módulos. El procesador p_1 intercambia datos con p_2 para calcular el número $q_1 * q_2$ en la primera fase con $d = 0$. La segunda fase es conseguir $q_3 * q_4$ de p_3 en $d = 1$. Finalmente, el número Q se calcula después de que los procesadores p_1 y p_5 intercambien datos, con $d = 2$.

Algoritmo 1 (Computación paralela de Q y Q_i***)***

```
Begin
  Forall processor p_i, 1 ≤ i ≤ m do in parallel
    Begin
      processor p_i : Q_i = k, temp' = Q = q_i;
    End
  For d = 0 ~ (log_2 m) − 1 do
    Begin
      Forall processor p_n, n = i + 2^(d+1) * j, where
        1 ≤ i ≤ 2^d, 0 ≤ j ≤ (m/2^(d+1)) − 1 do in parallel
        Begin
          processor p_n : SenMsg(p_(n+2d)modm, Q),
                          ReceiveMsg(p_(n+2d)modm, temp');
          processor p_(n+2d)modm : SenMsg(p_n, Q),
ReceiveMsg(p_n, temp');
        End
      Forall processor p_i, 1 ≤ i ≤ m do in parallel
```

Begin
Forall processors p_i :
$Q_i = Q_i * temp', Q = Q * temp', temp'' = Q$;
End
End
End

El Algoritmo 2 se usa para calcular el inverso multiplicativo de $Q_i \ mod \ q_i$, basándose en el algoritmo de Euclides para el cálculo del Máximo Común Divisor GCD. El GCD de dos números es uno si estos números son primos relativos. Por tanto, este algoritmo es una modificación del algoritmo de Euclides y funciona modulando estos coeficientes multiplicativos en el cálculo del GCD. Existe la condición de que los números Q_i y q_i deben ser coprimos a la entrada del algoritmo, debido a que este está basado en la característica comentada anteriormente para obtener el número b para encontrar $Q_i * b = r * q_i + 1$.

Algoritmo 2 (Inversa multiplicative $Q_i \ mod \ q_i$)

Input: Q_i, q_i
Output: b_i
Condition: $GCD((Q_i/k), q_i) = 1$
Begin
$g_0 = q_i, g_1 = (Q_i/k) mod \ q_i, v_0 = 0, v_1 = 1$;
Loop
If $g_1 \neq 0$ then $y = g_0 \ div \ g_1, g_2 = g_0 \ mod \ g_1$
else $b_i = v_0$, output b_i and halt;
$v_2 = y * v_1$;
While $v_0 < v_2$ do $v_0 = v_0 + q_i$;
$v_2 = v_0 - v_2, g_0 = g_1, g_1 = g_2, v_0 = v_1, v_1 = v_2$;
EndLoop
End

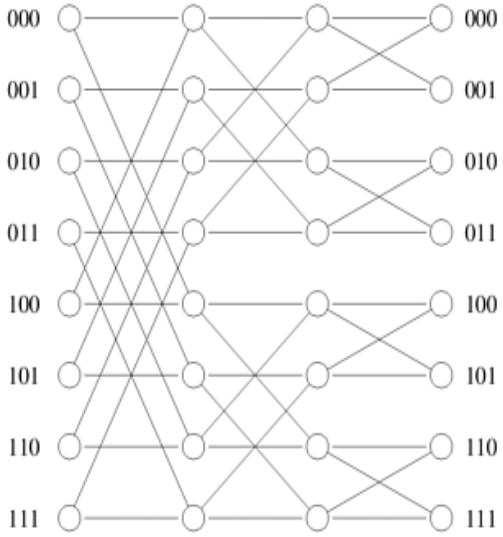
El Algoritmo 3 es el algoritmo final, que combina los Algoritmos 1 y 2 para computar el GCRT.

Algoritmo 3 (Computación de X)

Begin
Forall processor p_i, $1 \leq i \leq m$ do in parallel
Begin
processor p_i : input q_i and k; output Q_i and Q via Algorithm 1;
End
Forall processor p_i, $1 \leq i \leq m$ do in parallel
Begin
processor p_i : input Q_i, q_i; output b_i via Algorithm 2;
processor p_i : compute $N_i = \lceil (x_i * q_i/k) \rceil$;
End
Forall processor p_i
Begin
processor p_i : compute $X_i = (b_i * N_i \ mod \ q_i) * Q_i$;
$X = (X + X_i) \ mod \ (k * Q)$;
End
End

Y todo esto diseñado para operar sobre una red multiprocesador de tipo mariposa, tal y como se muestra en el siguiente diagrama.

Fig. 1. Esquema de red de mariposa usado en la arquitectura base del sistema.



Los primeros problemas que podemos observar de esta solución propuesta, son los derivados de las necesidades hardware. En concreto, necesita de una red de multiprocesadores con una arquitectura determinada para su correcto funcionamiento. Esto no es algo trivial, e impone un problema a la hora de su implantación, pues necesitaremos de un equipo que cumpla estas características a rajatabla, o en su defecto una capa de emulación de dicha red sobre el equipo.

En cuanto a los problemas derivados de la eficiencia y la gestión de memoria, un enfoque como este sobre una arquitectura monoprocesador sería impensable debido a los problemas de falta de espacio en memoria. Esto, según los propios autores ([LCh]), se puede solucionar debido a la gestión de la memoria conjunta de los m procesadores usados en el proceso de computación, pero no aportan resultados que apoyen esta idea.

Por tanto es una solución que solventa los problemas derivados de la eficiencia y gestión de memoria propios de una paralelización de GCRT, pero presenta unos requisitos de implementación demasiado exigentes, en tanto en cuanto se busque su implementación en sistemas no de alto rendimiento.

2) Parallel computational of Residue Number System – 2006 [ChKL]

Siguiendo la tónica de la referencia anterior, se propone una implementación software del teorema chino de los restos basada en la potencia de una arquitectura multiprocesador de ejecución paralela.

En este caso se usa una topología de anillo, necesitando de $2m$ procesadores, donde m es el número de congruencias del sistema de congruencias proporcionado.

Al igual que en [LCh] la implementación y adaptación del mencionado sistema a una arquitectura multi-hilo en un mono-procesador conllevaría un uso y gestión de recursos que caerían en la total ineficiencia del sistema.

Por tanto, concluimos igualmente que esta solución, aunque efectiva, no está al alcance de sistemas de media o baja gama, requiriendo de una inversión considerable para poder llevar a cabo su desarrollo.

2. Implementaciones GPU

No se ha podido establecer ninguna fuente que presente algún tipo de técnica o idea, así como resultados y conclusiones, de la implementación del CRT en entornos GPU.

Esto puede deberse en parte a los esfuerzos dirigidos en los últimos tiempos a las plataformas multi-procesador, que acaparan el entorno de soluciones del problema, y al abandono del CRT por parte de las soluciones criptográficas debido a las necesidades temporales resultantes de la aplicación del teorema.

Resumen

En resumen, podemos concluir que las arquitecturas hardware otorgan grandes capacidades de resolución al problema en sí, pero que su falta de adaptabilidad y escalabilidad hacen necesaria la búsqueda de una solución software que permita su extensión y utilización.

En esta vía, vemos como los esfuerzos se han dirigido hacia plataformas de computación de altas prestaciones y arquitectura multiprocesador, comunes en el ámbito científico, lo que provoca una desconexión entre las investigaciones y la aplicabilidad real de las soluciones estudiadas.

Por tanto, se hace necesaria una estructura software bien diseñada, e implementada, que permita la escalabilidad, para potenciar las deficiencias hardware, y adaptativa, de tal forma que el rendimiento aumente paulatinamente, siguiendo el avance de la mejora ofrecida por el avance de la arquitectura sobre la que se asiente.

Con estas premisas, se pretende hacer frente a las deficiencias de las soluciones estudiadas con anterioridad y presentadas aquí, buscando obtener un sistema de eficiencia suficiente, sobre arquitecturas comunes en el paradigma computacional de hoy día, y adaptado al nuevo modelo de procesamiento que poco a poco se empieza a imponer en paralelismo.

D. Solución Aportada

A continuación explicaremos las dos alternativas paralelas inicialmente concebidas, a fin de presentar de forma teórica que se piensa llevar a cabo, para en el siguiente capítulo, dedicado a la implementación, explicar lo que se ha hecho para implementarlas, y que se ha tenido que hacer para adaptarlo al paradigma GPU.

Para llevar a cabo lo descrito anteriormente, es necesario establecer una solución al CRT paralela usando los recursos disponibles en un PC convencional, de gama media-alta. De esta manera, se consigue la generalización de la solución al mayor número de arquitecturas posible.

Tras el análisis de las fuentes mencionadas anteriormente y de múltiples recursos acerca del CRT, se establecen dos posibles vías o puntos de paralelización en el algoritmo, susceptibles de ser explotados sin comprometer la fiabilidad del mismo.

Procedemos a continuación a detallar ambas vías, así como la motivación de las mismas y su posible impacto en la eficiencia del algoritmo.

Paralelización por Fuerza Bruta

En esta opción, vamos a presentar un posible diseño del algoritmo que aproveche una zona muy concreta del mismo que se realiza de forma secuencial y no contiene dependencias, para lanzarla a ejecución paralela.

Para construir una solución simultánea, primero tenemos

$$M_k = m/m_k$$

Para $k = 1, 2, \dots, n$. M_k es el producto de los módulos excepto m_k. Dado que m_i y m_k no tienen factores en común mayores que 1 cuando $i \neq k$, quiere decir que $gcd(m_k, M_k) = 1$. En consecuencia, dado el Teorema 1, sabemos que existe un entero y_k, inverso de M_k modulo m_k, tal que

$$M_k y_k \equiv 1 (mod\ m_k)$$

Para construir la solución simultanea, formamos la suma

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots + a_n M_n y_n$$

Si observamos de nuevo al demostración del teorema, podemos encontrar fácilmente esa zona que queremos explotar de forma directa.

Para construir una solución simultánea, primero tenemos

$$M_k = m/m_k$$

Para $k = 1, 2, \dots, n$. M_k es el producto de los módulos excepto m_k

---

En primer lugar, vemos que se calcula un nuevo módulo para cada congruencia del sistema, de forma totalmente secuencial, calculando uno cada vez, y sin existir dependencia en estos cálculos, luego hemos encontrado un primer punto donde podemos aplicar nuestra estrategia sin mayores problemas.

Dado que m_i y m_k no tienen factores en común mayores que 1 cuando $i \neq k$, quiere decir que $gcd(m_k, M_k) = 1$. En consecuencia, dado el Teorema 1, sabemos que existe un entero y_k, inverso de M_k modulo m_k, tal que

$$M_k y_k \equiv 1 (mod\ m_k)$$

Posteriormente se calcula el inverso del nuevo módulo, módulo el módulo anterior, y esto, otra vez, para cada congruencia, de forma independiente de las demás. Hemos localizado otro punto interesante de paralelización.

Para construir la solución simultanea, formamos la suma

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots + a_n M_n y_n$$

Ahora demostramos que x es una solución simultánea.

Finalmente, se realiza la construcción de una suma, cuyos sumandos con independientes, uno por congruencia, en base a los parámetros anteriormente calculados. Luego podemos establecer que el cálculo de los sumandos puede ser realizado de forma paralela e independiente.

Por tanto, y revisando lo que hemos estado obteniendo de la demostración, para realizar el proceso de la obtención de la solución del sistema de congruencias, según el teorema chino de los restos, debemos realizar una serie de pasos únicos, sobre cada congruencia, para, una vez finalizado todo el cálculo sobre las congruencias de forma independiente y construidos los sumandos, realizar una suma y un módulo, de forma secuencial.

Ahora, con estos puntos importantes, vamos a establecer que cambios debemos introducir a una implementación secuencial del CRT para conseguir las paralelizaciones buscadas.

CRT Secuencial

```
Begin
  For 1 ≤ i ≤ n do
    Begin
      m = m * m_k
    End
  For 1 ≤ i ≤ n do
    Begin
      M_k = m/m_k
    End
  For 1 ≤ i ≤ n do
    Begin
      y_k = M_k inverse m_k
    End
  For 1 ≤ i ≤ n do
    Begin
      s_k = a_k M_k y_k
    End
  For 1 ≤ i ≤ n do
    Begin
      S += s_k
    End
  S = s mod m
    End
```

Como podemos ver, se basa en una consecución de bucles de repetición sobre las tareas que hemos comentado anteriormente, con lo que vamos a marcar los bucles en los que hemos estudiado de manera teórica sobre el teorema y su demostración que no se producen dependencias, y comprobamos que esto es correcto.

CRT Secuencial

```
Begin
  For 1 ≤ i ≤ n do
    Begin
      m = m * m_i
    End
  For 1 ≤ i ≤ n do
    Begin
      M_i = m/m_i
    End
  For 1 ≤ i ≤ n do
    Begin
      y_i = M_i inverse m_i
    End
  For 1 ≤ i ≤ n do
    Begin
      s_i = a_i M_i y_i
    End
  For 1 ≤ i ≤ n do
    Begin
      S += s_i
    End
  S = s mod m
End
```

Así, quedan estos tres bloques de repetición marcados, como las tres zonas susceptibles de paralelización anteriormente señaladas. Podemos comprobar cómo, efectivamente, se realiza un procesado totalmente independiente, ya que cada etapa del bucle se procesa solo la variable afectada por la iteración actual, y esta no es usada en ninguna iteración distinta.

Además, podemos también observar que, estos tres bloques marcados, iteran sobre el mismo conjunto y en el mismo orden, y los resultados de la iteración x del primer bucle, son usados en la iteración x de los siguientes bucles única y exclusivamente, por lo que podemos simplificar la estructura del diseño de la siguiente manera.

CRT Secuencial

```
Begin
  For 1 ≤ i ≤ n do
    Begin
      m = m * m_i
    End
  For 1 ≤ i ≤ n do
    Begin
      M_i = m/m_i
      y_i = M_i inverse m_i
      s_i = a_i M_i y_i
    End
  For 1 ≤ i ≤ n do
    Begin
```

S += s_i
End
$S = s \bmod m$
End

Con esto, acabamos de solventar el problema de la paralelización de las tres zonas de forma independiente, reduciendo todo a un único bucle de repetición, donde en cada iteración se realizan tres acciones, dependientes entre si, por lo que su ejecución será secuencial, pero independientes entre las iteraciones del mismo bucle, donde aplicaremos el paralelismo en esta ocasión.

Así, de esta manera, resulta la siguiente modificación del algoritmo anterior, introduciendo paralelización.

CRT Paralelo Fuerza Bruta

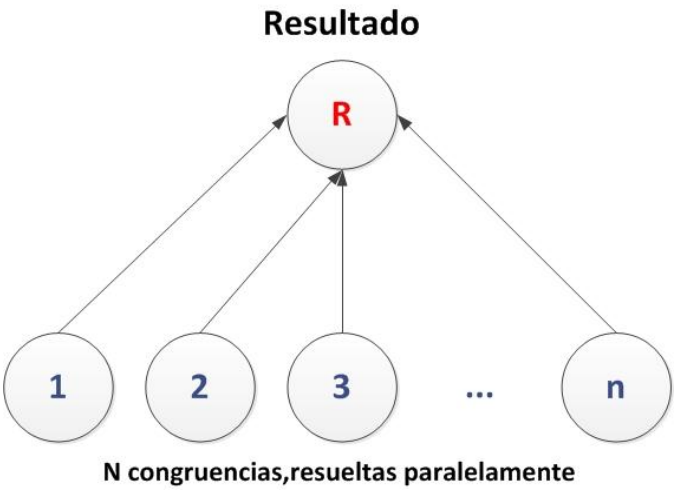
Begin
For $\mathbf{1} \leq i \leq n$ do
Begin
$m = m * m_i$
End
Forall processor p_i, $\mathbf{1} \leq i \leq n$ do in parallel
Begin
$M_i = m/m_i$
$y_i = M_i \ inverse \ m_i$
$s_i = a_i M_i y_i$
End
For $\mathbf{1} \leq i \leq n$ do
Begin
S += s_i
End
$S = s \bmod m$
End

En nuestro caso, cada procesador p_i se corresponderá con un hilo en nuestro sistema multi-hilo, para huir así, como buscamos, de arquitecturas cerradas multiprocesador, y posibilitando la escalabilidad, al no depender del número de procesadores.

Por tanto, nuestro nuevo bucle de tareas sobre congruencias realizaría tres acciones distintas, relacionadas entre sí y de manera secuencial, pero cada iteración se realizaría de forma paralela a las otras.

El siguiente esquema representa como se resolvería el problema y se daría solución a un sistema de n congruencias, en el caso ideal, donde los recursos son suficientes para realizar una ejecución paralela de todas las iteraciones del bucle.

Fig. 2. Diagrama de ejecución del diseño de paralelización por Fuerza Bruta.



Si nuestro sistema presenta una arquitectura multiprocesador, como en las referencias usadas, cada procesador podría realizar una o varias iteraciones del bucle, pero si contamos con un pc mono-procesador, multi-núcleo o mono-núcleo, se crearán múltiples instancias, una por iteración del bucle, que serán atendidas por los procesadores o núcleos en el momento en que alguno de ellos este libre. Así, cumplimos el primer objetivo de escalabilidad, pues la ejecución se adapta al número de unidades disponibles, hasta el límite, n, de congruencias del sistema. De igual manera, este nuevo diseño es adaptable, concepto de adaptabilidad, a cualquier arquitectura presente en sistemas de computación actuales, eliminando la necesidad de una arquitectura específica para su funcionamiento.

Es importante mencionar que el rendimiento y la eficiencia del diseño presentado, dependerá en gran medida de las prestaciones y la arquitectura del sistema sobre el que se decida llevar a cabo la implementación, generando así mejores resultados a mayor número de unidades de procesamiento disponibles.

Además, damos solución a un problema de escalabilidad en la variación del tamaño del problema, al no necesitar ya la adición de nuevas unidades de procesamiento, para poder llevar a cabo la resolución de un problema de dimensión mayor a la implementada y tenida en cuenta en el desarrollo de la arquitectura, siendo en este caso totalmente dinámico, y solo limitado por los recursos de memoria del equipo y tiempo de cómputo disponible.

Limitaciones

Aun teniendo en cuenta todas las ventajas introducidas por este nuevo diseño, es importante destacar también las limitaciones que, a priori, se pueden observar en su concepción.

A pequeños tamaños de los sistemas de congruencias objetivo, no se prevén mayores dificultades, salvo la gestión de las unidades de procesamiento compartidas, problema

previsto y solucionado en la mayoría de plataformas de implementación paralelas. Pero cuando el tamaño aumenta de forma considerable, o el tamaño de los enteros manejados crece, podemos incurrir en problemas de espacio en la memoria del sistema.

Esto es así, debido a que, aunque trabajamos de forma paralela, hay un paso inicial en el algoritmo que es importante recordar.

```
For 1 ≤ i ≤ n do
  Begin
    m = m * m_i
  End
```

En este paso, realizamos una multiplicación de todos los módulos de todas las congruencias del sistema de congruencias, para formar un módulo mayor. Si manejamos un gran número de congruencias, o trabajamos con enteros largos, caso común en criptografía, podemos llegar a tener un problema en la gestión de memoria debido al gran tamaño que las variables que vamos a manejar van a alcanzar, por lo que se hace necesaria la proposición de otra alternativa que solvente de alguna manera los problemas originados en este paso inicial.

Paralelización por Eficiencia

Con el fin de solventar los posibles problemas de gestión de memoria descritos en el diseño anterior, se propone un nuevo diseño basado en una división eficiente del problema, siguiendo un esquema del tipo Divide y Vencerás, a fin de evitar el trabajo con números grandes en los pasos iniciales, y retrasar lo máximo posible su manejo. Notemos que la corrección de esta propuesta viene dada por la demostración del Teorema 2.

Buscamos pues dividir el problema de forma recursiva hasta un caso base de tamaño predeterminado, momento en el cual se procede a la resolución del problema, propagando los resultados desde las ramas hasta la raíz sobre el árbol de resolución generado en el proceso de división del sistema de congruencias.

Cada caso resuelto, arrojará una nueva ecuación en congruencias que, propagada hacia el nodo padre en el árbol, y compuesta con los resultados extraídos de sus nodos hermanos, genera un nuevo sistema de congruencias, resultando pues una aplicación sucesiva y por grupos y niveles de la resolución por medio del teorema chino del resto.

De forma general, el algoritmo que define esta opción de diseño es el que aparece a continuación.

CRT Eficiencia

```
Begin
  If Size ≤ Limit then
    CRT (Parallel or Secuential)
  else
    CRT Eficiencia(First)
    CRT Eficiencia(Second)
    Compact Solution
  end if
End
```

En este algoritmo se establece un límite, o caso base, que será el número máximo de congruencias que ha de tener un sistema para ser resuelto de forma directa, y dejar de subdividir el problema.

De esta manera, se puede establecer un caso baso de forma dinámica, según los recursos del sistema anfitrión, y usar como resolución una versión secuencial del cálculo del CRT, o la versión paralela descrita anteriormente, ya que trabajaremos con conjuntos más reducidos y así evitamos los problemas de gestión de números enteros demasiado grandes.

Además, se introduce un diseño básico, en el que el conjunto se subdivide en 2 porciones, resultando un nuevo análisis de cada una de ellas por el mismo algoritmo, pero, si es estima oportuno y el conjunto así lo posibilita, se pueden establecer distintas subdivisiones, siendo posible crear tres, cuatro o más subconjuntos de forma simultánea, y estudiarlos todos.

Con el fin de introducir un grado de paralelización en este algoritmo, además del posible uso del CRT por Fuerza Bruta en la resolución directa, se puede realizar la siguiente modificación al algoritmo.

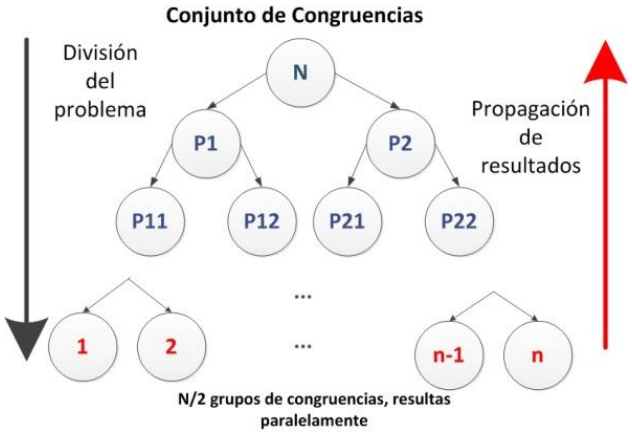
CRT Eficiencia

```
Begin
  If Size ≤ Limit then
    CRT (Parallel or Secuential)
  else
    do in parallel
      CRT Eficiencia(First)
      CRT Eficiencia(Second)
     end do
    Compact Solution
  end if
End
```

Y de esta manera, todas las subdivisiones que se generen en cada iteración del algoritmo, se lanzarán, ahora sí, de forma simultánea a exploración, generando una mayor eficiencia y un mayor aprovechamiento de las unidades y capacidades de procesamiento disponibles.

Para ilustrar este nuevo diseño y su comportamiento, podemos observar el siguiente diagrama.

Fig. 3. Diagrama de ejecución del diseño de paralelización por eficiencia.



Como podemos ver, partimos de la raíz del árbol con un conjunto de congruencias, que forman nuestro sistema de congruencias inicial, y este se va subdividiendo, para el caso natural de dos divisiones por análisis, hasta la raíz, donde el tamaño es suficientemente pequeño para ser resuelto de forma directa sin problemas.

Una vez resuelto en la raíz, los resultados son propagados de forma ascendente por el árbol en forma de ecuación en congruencias, así, en el padre se forma un nuevo sistema de congruencias, que es resuelto y elevado en forma de nueva ecuación, así hasta el nodo raíz, donde se realiza la última resolución de sistema de congruencias y se genera el resultado.

De esta forma, resolvemos a priori las peculiaridades que puedan ocurrir debido al trabajo reiterado con Enteros Grandes y postergamos el trabajo con estos datos hasta la resolución de las capas altas, y nunca con más datos que número de subdivisiones hagamos por conjunto, generando así operaciones más limpias, rápidas y seguras.

Limitaciones

En este caso, perdemos el factor de alta paralelización logrado en el diseño anterior, a cambio de estabilidad y eficiencia en el procesado de los datos y la consecución de la solución.

A priori, las limitaciones se establecen en el mismo punto que cualquier algoritmo de carácter recursivo, en el uso de recursos de memoria. Aun así, no es un factor preocupante, ya que no hay una generación de datos continua, si no que se trata de un conjunto inicial de valores, del cual se genera una cantidad de nuevos datos limitada en comparación al tamaño del conjunto.

La mayor limitación a priori es el tiempo en la capacidad de computo de las unidades de procesamiento. Al introducir la tarea de subdivisión en el diseño, se espera un comportamiento lento a pequeñas cantidades de datos, comportamiento que se espera estabilizará cuando el conjunto de datos aumente, generando una mayor eficiencia en los procesos de subdivisión y compactación de datos.

III. IMPLEMENTACIÓN

En este apartado, se van a presentar todos los detalles de implementación de los diseños anteriormente expuestos, así como las necesidades de adaptación, en caso de ser necesario, de los mismos ante los distintos lenguajes de desarrollo planteados como objetivo.

Para llevar a cabo esta implementación se han usado dos vías de programación paralela, a fin de comparar el comportamiento de los algoritmos ante diversos enfoques de ejecución simultánea. En concreto, haremos uso de C# y el conjunto de librerías Parallels .Net de Microsoft, y por otro lado, daremos un enfoque más novedoso usando CUDA y C++ para extraer resultados y poder compararlos.

Además, ha sido necesaria la implementación de ciertos algoritmos para generar el conjunto de datos de entrada del CRT, comunes a ambas plataformas, por lo que comenzaremos por dar cuenta de ellos.

A. Generación de Datos de Entrada

Es necesario contar con un buen conjunto de datos de entrada a la hora de implementar y probar el correcto funcionamiento de los algoritmos diseñados anteriormente, por lo que se hizo necesario el contar con dos procesos auxiliares para este cometido.

En primer lugar, se hace necesario el desarrollo de un generador de números aleatorios. Estos números, deben ser aleatorios y se convertirán en los coeficientes de las ecuaciones en congruencias que formarán el sistema de congruencias entrada del CRT. Además, deben estar comprendidos en un cierto, donde se asegure que han de tener los bits necesarios especificados, a efectos de utilización en sistemas criptográficos, además de no superar ni igualar al número que hará las veces de módulo en su misma ecuación en congruencias.

De igual manera, necesitaremos, como hemos dicho, un conjunto de números que harán el papel de módulos en las ecuaciones en congruencia. Estos deben ser, además, primos relativos entre sí, y serán un bit mayores que los coeficientes de las ecuaciones, para evitar el caso de que estos sean menores que aquellos.

Para el caso de generación de números primos se probaron varias alternativas y varios algoritmos conocidos, pero se tornaban ineficientes conforme el conjunto de datos crecía, así que se optó por implementar una versión del algoritmo de comprobación rápida de primos de Miller-Rabin. Conocido como el test de primalidad de Miller-Rabbin ([R]), se trata de un algoritmo determinista, basado en la hipótesis generalizada de Riemann ([B]). Más tarde, Rabin lo modificó a fin de obtener un algoritmo probabilístico incondicional.

Test de Primalidad de Miller-Rabin

Supóngase que $n > 1$ es un número impar del cual queremos saber si es primo o no. Sea m un valor impar tal que $n - 1 = 2^k m$ y a un entero escogido aleatoriamente entre 2 y $n - 2$.

Cuando se cumple

$$a^m \equiv \pm 1 \; mod n$$

o bien

$$a^m \equiv \pm 1 \; mod n$$

Para al menos un r entero entre 1 y $k - 1$, se considera que n es un primo probable. Si ha resultado primo probable, se escoge un nuevo valor para a, y se itera nuevamente reduciendo el margen de error probable.

Además, asumiendo correcta la hipótesis de Riemann, se puede demostrar que, si todo valor de a hasta $2(\ln n)^2$ ha sido verificado y n todavía es clasificado como primo probable, entonces n es un número primo.

Para el caso de la generación de números, se ha optado por el uso de un generador aleatorio de números, o de un generador de números consecutivos a partir de un número mínimo de bits.

A efectos de eficiencia en pruebas de corrección de los algoritmos se usará el generador consecutivo de números. Para este caso también se han encontrado ciertas complicaciones. En primer lugar nos encontramos con el hecho de que trabajaremos con enteros largos, por lo que es necesario establecer cuál va a ser el tipo elegido para su representación.

En la versión implementada de trabajo sobre C#, se tomaron dos alternativas interesantes. Por un lado, la implementación nativa correspondiente a BigInteger proporcionada por el lenguaje, y por otro lado, a fin de obtener mayor capacidad en nuestros enteros largos, se optó por la librería externa GNU MP, Gnu Multiple Precision, adaptada recientemente a C# y cuyo uso está bastante extendido en aplicaciones que requieren de estos tipos en lenguaje C y C++.

Esto condicionará, además la existencia de distintas versiones de los mismos algoritmos, haciendo uso de las funciones de adición, multiplicación, exponenciación y división de las distintas opciones escogidas.

Para el caso de implementación en CUDA y C++, surgió un gran problema de base, relativo al uso de BigInteger. CUDA no soporta de forma nativa este tipo de enteros y, además, versiones relativamente antiguas de su soporte hardware solo son capaces de trabajar con aritmética de 32 bits. Además, no existe aún una conversión de la librería GNU MP para esta plataforma, por lo que estamos forzados a usar aritmética nativa o aritmética de enteros largos propia.

Una vez definidos estos dos pequeños algoritmos, pasaremos a definir la implementación de los distintos diseños mencionados de CRT paralelo.

B. Implementación de Diseños CRT Paralelo

A continuación vamos a presentar las distintas implementaciones realizadas de los dos diseños anteriores, dividiendo el contenido en las relativas a C# y las relativas a CUDA, para posteriormente realizar unas pequeñas pruebas de validación y verificación de los diseños implementados.

Implementación sobre C# y Parallels .Net

Para la implementación de ambos diseños sobre C# se ha usado el entorno de desarrollo Microsoft Visual Studio 2010 y la versión de compilación del Framework de .Net 4.0. Además, se ha hecho uso de la librería externa GNU MP 1.0, compilada sobre .NET 2.0 pero totalmente funcional y estable sobre la versión 4 del Framework.

La decisión de usar GNU MP sobre C# recae en la gran aceptación obtenida en el campo de los enteros largos, y se ha hecho uso de un wrapper para C# que posibilita que este lenguaje comienza a dar sus pasos sobre el campo de los enteros largos de la mano de GNU MP, y así comprobar su eficiencia real frente a la implementación nativa de Microsoft que ofrece bajo el nombre BigInteger, dentro del paquete Numerics.

El uso del lenguaje C# viene condicionado por uno de los objetivos inicialmente buscados, la máxima integración posible sobre nuevas arquitecturas. Actualmente, .NET es la base de un gran número de desarrollos PC y Web que se ejecutan sobre plataformas Windows, convirtiéndose además en un lenguaje, C#, que interactúa de forma muy eficiente con el sistema operativo Windows haciendo un uso inteligente de los recursos, sobre todo en su versión paralela. Además, se ha convertido en una plataforma casi portable al realizarse los desarrollos paralelos de Mono, C# sobre Linux, a fin de posibilitar una portabilidad de las aplicaciones mucho menos costosa. Tanto es así que se está usando incluso en dispositivos de juego portátil, como PS VITA, como lenguaje de desarrollo de aplicaciones para el mismo, lo cual da una idea de su impacto en la tecnología actual y sus posibilidades.

El motivo principal del uso de Parallels .NET y no otras librerías de programación multihilo sobre C#, es el procesado interno que acompaña a la llamada de una instrucción paralela como Parallel.For. Por defecto, usa la cola de trabajos del ThreadPool del Framework de .NET para ejecutar el bucle, y usar el mayor paralelismo posible según el uso de recursos del sistema de forma automática. Además, proporciona ciertos mecanismos de seguridad y eficiencia interesantes, como son el manejo de excepciones en threads del pool de hilos corriendo el bucle lanzado, donde una excepción en uno de ellos origina que los demás sean llamados a finalizar su procesamiento lo más pronto posible, pero no cortados de manera drástica.

Entre otros es importante mencionar el control y coordinación de hilos y recursos en paralelización anidada, útil cuando procesos paralelos ejecutan tareas paralelas u otros procesos de paralelización directa. Además realiza un balanceo de carga eficiente y dinámico, gestionando el uso de los hilos hardware disponibles y actuando para evitar

situaciones de inactividad quedando trabajo restante, o largas colas de trabajo asociadas a uno de los recursos. De esta manera, se convierte en una gran opción de desarrollo a efectos de estabilidad y eficiencia sobre plataformas y arquitecturas variables, logrando otro de los objetivos propuestos.

Comenzaremos pues con los pormenores del algoritmo de paralelización por fuerza bruta, y continuaremos posteriormente con el de eficiencia por divide y vencerás.

1. Fuerza Bruta

La implementación del algoritmo de resolución de sistemas de congruencias mediante el CRT paralelizado por Fuerza Bruta se realizó siguiendo el diseño anteriormente presentado.

Debido a que contamos con dos implementaciones posibles para enteros largos, se desarrollan de forma paralela dos métodos que realizarán el procesamiento indicado en el diseño, pero bajo tipos de enteros largos distintos. Anteriormente se han definido los métodos necesarios para generar los datos que servirán de entrada para los diseños implementados, y en base a ellos se generan dos estructuras que usaremos aquí.

En primer lugar, contamos con un vector de coeficientes, que presenta los coeficientes de las **n** ecuaciones en congruencias. Por otro lado, contamos con otra estructura de tipo vector de módulos, donde presentaremos los módulos de las mismas congruencias, y realizando una paridad entre ambos vectores. Por tanto en la primera posición del vector, encontramos los valores representativos de coeficiente y módulo de la primera congruencia, y así sucesivamente hasta **n**.

Siguiendo el algoritmo propuesto, en primer lugar debemos generar un nuevo módulo, producto de los módulos de todas las congruencias. Esta tarea se realiza de forma secuencial. A continuación es donde se realiza el procesamiento paralelo propiamente dicho. En este caso haremos uso de la instrucción Parallel.For que genera un bucle de repetición de iteraciones conocidas, donde cada iteración se realiza de forma simultánea con las demás.

A tal efecto se establece como delegado el parámetro i, que será el índice de diferenciación entre los hilos, y se corresponde con cada una de las posiciones de los vectores anteriormente mencionados.

En primer lugar se realiza el cálculo de un nuevo módulo de la congruencia, en función del cálculo secuencial anterior, y se procede a la siguiente tarea.

Ahora, ha de realizarse el cálculo del inverso en módulo del nuevo módulo de la congruencia frente al nuevo módulo calculado de forma secuencial. Para el cálculo del multiplicativo inverso comentado, se hace uso de una versión del Algoritmo Extendido de Euclides, que igualmente tendrá una versión sobre BigInteger y otra sobre GNU MP.

Una vez realizada esta tarea, se pasa a generar el sumando que será el valor resultado de todo el proceso al finalizar la iteración del bucle, donde se realiza la multiplicación del coeficiente, inverso y módulo calculados para la congruencia actual. Todos estos valores de cálculo y resultados son almacenados en posiciones de vectores, uno por valor calculado, cuyos índices coinciden con el índice de la congruencia actual en los vectores de coeficientes y módulos.

Por último, se realiza la suma de todos los sumandos calculados de forma secuencial, y el resultado se define módulo el resultado del primer paso del algoritmo, obteniendo el resultado del sistema de congruencias lineal aportada a la entrada del algoritmo.

2. Eficiencia

La implementación del algoritmo de resolución de sistemas de congruencias mediante el CRT paralelizado por Eficiencia se realizó siguiendo el diseño anteriormente presentado. En este caso se decidió no implementar las versiones de BigInteger nativo y GNU MP, por lo que se implementará solo la versión que hace uso de los enteros largos nativos.

Debido a que se trata de una metodología divide y vencerás, tanto los parámetros de entrada del algoritmo como su estructura difieren considerablemente del anterior.

En este caso, se necesitaran los dos vectores anteriormente explicados de coeficientes y módulos de las ecuaciones del sistema de congruencias, además de dos parámetros extras, que representan el inicio y el final del subconjunto a explorar por el algoritmo. Estos últimos parámetros responden a una cuestión de eficiencia de computación. Puesto que el enfoque divide y vencerás exige una subdivisión continua en conjuntos hasta llegar al caso base, se ha tomado como decisión que las divisiones serán totalmente virtuales, y todos las ejecuciones del algoritmo realizadas trabajarán sobre el mismo conjunto de datos compartido, evitando así consumos de memoria excesivos y tiempos de división de conjuntos y copia de variables.

Para comenzar, siguiendo el esquema Divide y Vencerás, se establece la condición de parada de la recursión en función del caso base establecido. Más adelante se establecerá el mejor caso base. En caso de darse un acierto en la condición, se realiza una llamada al algoritmo de paralelización por Fuerza Bruta modificado levemente.

Usamos el algoritmo anterior debido a que intentamos resolver un caso base, por lo que debe ser un caso forzosamente de tamaño reducido, donde, teóricamente, el diseño anterior debería funcionar con un alto grado de paralelismo y eficiencia sin contratiempos.

Las modificaciones realizadas atañen a los parámetros de entrada, donde de nuevo debemos incluir los límites del conjunto que debe usar el algoritmo para realizar la resolución directa. Además, se modifica la salida del mismo, ya que no buscamos simplemente una solución al sistema, si no la clase de equivalencia que da solución al sistema, es decir, tanto el coeficiente como el módulo generados en el proceso.
En caso de no darse la condición, comienza el procesamiento del algoritmo propiamente dicho.

Calculamos el tamaño de las subparticiones a generar del conjunto de datos definido, y procedemos a la llamada recursiva a este mismo algoritmo. Es aquí donde realizamos la llamada a la misma instrucción Parallel.For de paralelización que en el algoritmo anterior. De esta manera, todas las

subdivisiones generadas se lanzarán a exploración a la vez, generando un alto grado de paralelismo, que crecerá conforme bajemos niveles en el árbol y la profundidad aumente.

Una vez terminado el procesamiento de los subconjuntos explorados, es necesario compactar las soluciones recogidas. Para ello guardamos los resultados obtenidos en dos vectores, uno relativo a los coeficientes y otro a los módulos de las ecuaciones en congruencias obtenidas, y llamamos a un algoritmo de resolución del CRT secuencial. El hecho de no usar en este caso un algoritmo paralelo se debe a que el número de subconjuntos va a ser demasiado pequeño para que un algoritmo de ejecución paralela resulte eficiente. Más adelante se detallarán estos tamaños.

Finalmente se devuelve el resultado en la forma de par "coeficiente-módulo". De esta manera, los resultados escalarán en el árbol de divisiones hasta llegar a la raíz, cuya devolución arrojará el resultado final, resolviendo así el sistema de congruencias lineales mediante el CRT.

Implementación sobre CUDA y C++

Para la implementación sobre CUDA y C++ se ha usado el entorno de programación Microsoft Visual Studio 2010, a fin de que los resultados de las ejecuciones en etapas posteriores no se vean afectados por diferencias en este aspecto. Se ha realizado la compilación sobre la versión 4.1 del SDK de CUDA.

El principal motivo de usar CUDA es el auge que parece sufrir como tecnología de paralelización en múltiples aplicaciones en el panorama computacional, (Ver [BBBC], [KR], [ZHChFYY], [MA]). Además se trata de una tecnología capaz de correr sobre cualquier dispositivo que contenga una tarjeta gráfica de la marca NVIDIA desde versiones de cierta antigüedad, ofreciendo la posibilidad al usuario de obtener potencia de una arquitectura multiprocesador y multicore desde su ordenador personal.

Si se extrapola el concepto hacia dispositivos orientados a la prestación de servicios, como servidores de contenido digital en streaming, que requieren de procesos de encriptación constantes y que han de ser transparentes al usuario, esta tecnología puede suponer un aumento considerable en su capacidad de procesamiento de peticiones a un coste relativamente bajo.

Se barajó la posibilidad de usar OpenCL en lugar de CUDA, por tratarse de un entorno abierto y portable a hardware de la competencia de NVIDIA, ATI, pero el rendimiento ofrecido no llega a los niveles que ofrece CUDA, por lo que se descartó inmediatamente.

En cuanto a la elección de C++ como lenguaje anfitrión del host y comunicación con CUDA, se debió a que este es el lenguaje recomendado por NVIDIA para realizar las implementaciones, además de presentar las mejores características de comunicación con CUDA SDK. Existen wrappers orientados a realizar uso del mismo SDK sobre otros lenguajes como C# o Java, pero fueron desechados por suponer la introducción de nuevas capas en la tarea de comunicación con el SDK.

Comenzaremos pues con los pormenores del algoritmo de paralelización por fuerza bruta, y continuaremos posteriormente con el de eficiencia por divide y vencerás, como en la sección anterior.

1. Fuerza Bruta

Partiendo del diseño del algoritmo se desarrolla la implementación, atendiendo con especial interés a la situación especial que impone el trabajar con dos lenguajes.

En este caso, todo lo que es susceptible de ser paralelizado, debe ejecutarse en contexto GPU, por lo que habrá de implementarse sobre CUDA, mientras que todo lo demás, se realizará sobre C++.

Los métodos de generación de los datos de entrada se mantienen en su misma versión, pero portados a C++. Además mantenemos el procesamiento basado en dos estructuras de tipo vector que guardarán los coeficientes y módulos de las ecuaciones en congruencias del sistema a resolver.

En primer lugar, debemos realizar una reserva de memoria en la GPU para los datos que va a manejar en su cómputo. Es decir, para los dos vectores de datos, y para un vector de resultados, al igual que en las implementaciones solo CPU.

Una vez realizados estos pasos previos, procedemos a calcular el módulo producto de los módulos de las ecuaciones del sistema, en C++ de forma secuencial, como hasta ahora.

A continuación, realizamos la llamada a la tarea CUDA que se ejecutará en la GPU, cuyos argumentos son los vectores de datos, el vector de resultados, el nuevo módulo y el número de elementos totales. En ésta, es necesario establecer el número de bloques por malla de procesadores, y el número de hilos que se ejecutarán por bloque. El número de hilos por bloque se establece típicamente a 256, pero puede ser cualquier múltiplo de dos, recomendable a partir de 64 o 128. En cuanto al tamaño de bloques por malla de procesadores, se establece como el entero mayor de la división del número total de elementos por el número de hilos por bloque, a fin de ajustar el número de recursos reservados y las ejecuciones a los datos de entrada. Este es un paso muy importante a la hora de implementar procesos en CUDA si queremos obtener buenos resultados de eficiencia más adelante.

El proceso CUDA se compone de las mismas acciones que el cuerpo del bucle de repetición paralelizado en la implementación host. En primer lugar se realiza el cálculo del módulo de la congruencia actual, para posteriormente calcular su inverso y generar el sumando. Estos sumandos se almacenan en el vector de resultados, que serán los usados para realizar la suma más adelante en C++. Una vez finalizado el procedimiento se realiza una sincronización de los procesos CUDA para comprobar la terminación adecuada y se mueven los resultados de la memoria de la GPU a la memoria de la CPU.

Una vez aquí, se realiza la suma y se aplica el módulo para obtener el resultado final. El proceso realizado es idéntico a las versiones CPU, solo que es necesario realizar particiones al método para extraer las partes susceptibles de paralelización y enviarlas a GPU.

2. Eficiencia

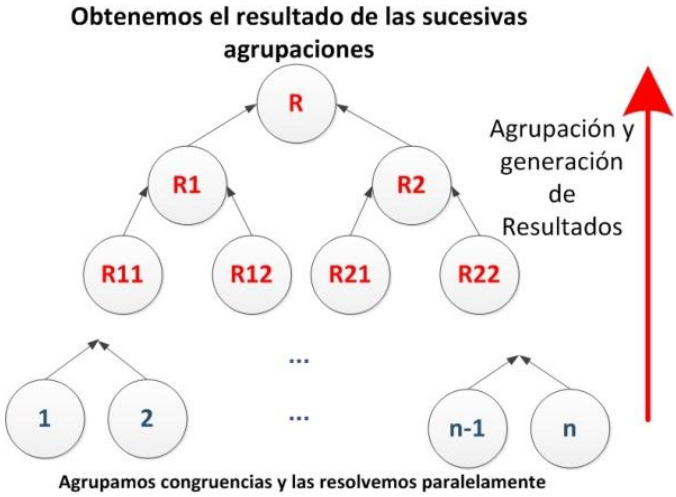
Para la implementación por eficiencia apareció un problema de magnitud considerable. Por su naturaleza, no es posible implementar un problema divide y vencerás recursivo, debido a que CUDA no acepta la recursión de ninguna de las maneras. Debido a ello, se hace necesario replantear el algoritmo y adaptarlo a la nueva situación.

Debido a que divide y vencerás recursivo no es aplicable a todas las arquitecturas, se hace necesario entonces una generalización del mismo, para ser adaptable a mayores ámbitos de implementación. Para ello se van a eliminar las llamadas recursivas, que además generan una gran carga computacional, en favor de estructuras de división más simples y eficientes.

En primer lugar se presenta la problemática de la división recursiva convertida en división directa, sin afectar de forma negativa al coeficiente de paralelismo. Este algoritmo parte de la premisa de que el conjunto de partida ya se encuentra dividido de forma eficiente. En nuestro caso práctico, esto se traduce en una deducción en base a las siguientes premisas: se pretende resolver un sistema de congruencias usando el CRT y el sistema de congruencias posee tamaño n. Por tanto tenemos a nuestra disposición tantas congruencias como el tamaño del sistema

Considerando esto, las alternativas son: Definir divisiones del conjunto de forma recursiva sobre CPU o Definir combinaciones de los elementos del conjunto (Asunción de división inicial). Por tanto, en base a un conjunto de entrada de congruencias, somos capaces de definir agrupamientos dentro del conjunto, eliminando el proceso de creación del árbol de resolución desde la raíz hasta las hojas, y comenzando la resolución y la creación del árbol directamente desde las hojas.

Fig. 4. Modificación del diagrama de paralelización por eficiencia generalizado.



Al igual que la implementación por Fuerza Bruta, en primer lugar reservamos espacio en la GPU y lanzamos los datos desde la memoria host a su memoria.

Una vez realizado esto, se procede a un número de llamadas sucesivas a la tarea CUDA que realizará el procesamiento paralelo, de forma consecutiva. El número de llamadas viene en función del logaritmo base 2 del número de elementos. Esto es así debido a que el proceso que vamos a seguir es el siguiente: desde el conjunto base de datos separados, agrupar de dos en dos para realizar una resolución del CRT para dos congruencias. Una vez realizado este proceso, el resultado se almacena, dando lugar a aproximadamente la mitad de nuevas congruencias que anteriormente. Este proceso se repite de forma secuencial hasta que el número de congruencias resultado sea 1, en cuyo caso hemos llegado a la solución.

De esta manera, sobre host, C++, se realizan sucesivas llamadas a un procedimiento CUDA, que realiza esta acción, hasta que se realizan $\log_2 n$ iteraciones, para n congruencias en el sistema de ecuaciones inicial. Este proceso se dedica a realizar una resolución completa del CRT para dos ecuaciones solamente. La clave de esto está en que, cada iteración lanzada desde host, se realizan tantas resoluciones simples del CRT como $n/2$.

Una vez terminado el proceso ya no es necesario realizar sumas ni ningún tipo de operación, pues la última iteración resolvió las dos últimas ecuaciones en congruencias ofreciendo el resultado del CRT para el sistema completo.

C. Validación y verificación de la implementación

Para la verificación de las implementaciones realizadas, se procedió en primer lugar a seleccionar un pequeño conjunto de sistemas de solución conocida, y se aplicaron sobre los distintos algoritmos implementados, incluido el secuencial, a fin de servir de medida de prueba para las siguientes verificaciones.

Una vez comprobado su correcto funcionamiento y la corrección de los resultados, se procedió a realizar pruebas con los algoritmos de generación de datos de entrada, hasta tamaños de 100 congruencias, a fin de comprobar la verificación.

Una vez comprobada, se establecieron nuevas pruebas hasta tamaños de 1000 congruencias, eliminando de la verificación la implementación secuencial por inviable en medida de tiempo, verificando todas las implementaciones y estableciendo el correcto funcionamiento de los sistemas tanto sobre CPU como sobre la combinación CPU y GPU.

IV. RESULTADOS

A continuación, se van a presentar todos los resultados obtenidos durante las pruebas realizadas sobre los dos diseños implementados de las dos maneras presentadas anteriormente.

A. Entorno de pruebas

El sistema sobre el que se realizarán las distintas pruebas mencionadas a continuación y los resultados obtenidos durante las mismas es el mencionado en la tabla siguiente.

Tabla1. Especificaciones del Sistema Host para la realización de los estudios de eficiencia de los algoritmos.

Sistema Host	
Placa Base	Asus P6T
Memoria Ram	Kingston DDR3 1333 Mhz 2GB x 2 (PC3-10700)
Procesador	Intel Core i7 920
Disco Duro	Seagate 1TB
Tarjeta Gráfica	NVIDIA Geforce GTX 260 – Asus ENGTX260 GL+
Sistema Operativo	Windows 7

En cuanto a las características propias de cada una de las unidades de procesamiento por separado, estos datos se recogen en las siguientes dos tablas presentadas.

Tabla 2. Especificaciones de la CPU mencionada en el sistema.

CPU	
Denominación	Intel Core i7 920
Consumo Máximo	130W
Tecnología	45 nm
Frecuencia de Trabajo	2.67GHz
Número de Núcleos	4
Número de Hilos por procesador	2
Número de Hilos totales	8
Frecuencia Máxima Turbo (Solamente un procesador)	2.93Ghz
Cache	8MB
Velocidad del Bus Frontal	4.8 GT/s
Ancho de Banda de memoria máximo	25.6 GB/s
Velocidad de Memoria	1066 Mhz (DDR3)
Interfaz de Memoria	36 bits

Tabla 3. Especificaciones de la GPU mencionada en el sistema.

Tarjeta Gráfica	
Denominación	Asus EN260GTX GL+
Consumo Máximo	182W
Tecnología	55 nm
Número de Procesadores Shader	216 unificados
Número de Multiprocesadores	27
Número de procesadores por multiprocesador	8
Frecuencia de Trabajo del motor	576 Mhz
Frecuencia de los procesadores Shader	1242 MHz
Ancho de Banda de memoria máximo	113.7 GB/s
Velocidad de Memoria	1999 Mhz (DDR3)
Tamaño Memoria	896 MB
Interfaz de memoria	448 bits

Como se puede observar a primera vista, se trata de un sistema de un equipo cuyas prestaciones en el momento actual se encuentran en una gama media. Las principales diferencias entre las unidades de proceso que se van a utilizar destacan en el número de unidades de proceso, su velocidad, y la velocidad del acceso a memoria básicamente. En el caso de la CPU, contamos con pocos núcleos e hilos hardware, pero su velocidad de procesamiento es mucho más alta que la de los procesadores GPU, que por otro lado son muchos más, ordenados en una arquitectura multiprocesador, y con un acceso a memoria mucho más rápido, con lo que se reducen las latencias enormemente.

Todas las pruebas van a ser realizadas sobre este sistema tomándolo como un entorno normal de trabajo. Es decir, no se han desconectado pantallas de la tarjeta gráfica para liberarla de trabajo, con el fin de no darle ventaja, y tampoco se han desconectado servicios innecesarios de Windows. Se trata de un arranque normal de un ordenador personal, donde las únicas aplicaciones corriendo en primer plano serán las necesarias para realizar las ejecuciones, pero manteniendo el entramado en segundo plano intacto. De esta manera se pretende emular el funcionamiento en un entorno real, donde difícilmente se dispondrá de un equipo de forma exclusiva.

B. BigInteger o GNU MP

En primer lugar, vamos a establecer en base a una batería de pruebas cuál es la implementación del tipo de datos entero largo que ofrece mejores resultados de eficiencia sobre C#, para usar su respectivo algoritmo en las pruebas sucesivas. De esta manera nos aseguramos contar con la mejor implementación de cada diseño sobre procesador convencional, y así conseguir unos resultados posteriores de enfrentamiento frente a procesador gráfico más veraces.

Para ello, se han lanzado 100 ejecuciones de cada implementación del algoritmo de paralelización por fuerza bruta para distintos valores entre 10 y 1000 congruencias, donde los módulos serán primos de 32 a 256 bits, y los coeficientes de 1 bit menos. Los resultados medios obtenidos pueden observarse en las siguientes tablas.

Tabla 4. Módulos de 32 bits en implementación por Fuerza Bruta.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.16	1.19
50	0.65	2.74
100	0.70	4.24
250	2.97	8.99
500	10.26	20.83
750	22.64	32.79
1000	39.74	51.10

Tabla 5. Módulos de 64 bits en implementación por Fuerza Bruta.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.24	0.93
50	1.39	2.43
100	2.84	4.13
250	13.45	7.77
500	47.86	19.33
750	106.02	32.13
1000	183.11	49.18

Tabla 5. Módulos de 128 bits en implementación por Fuerza Bruta.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.44	0.76
50	2.07	3.15
100	6.20	2.47
250	34.88	8.85
500	127.17	19.99
750	273.32	34.59
1000	474.52	51.64

Tabla 7. Módulos de 256 bits en implementación por Fuerza Bruta.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.61	1.23
50	5.46	2.57
100	18.63	4.17
250	96.57	8.69
500	364.06	21.31
750	786.85	34.27
1000	1402.83	51.70

Como podemos observar, para tamaño de enteros por encima de los 32 bits, y a partir de un número de congruencias superior a 50 o 100, la implementación basada en GNU MP presenta mejores resultados. Además debemos contar con que las aplicaciones criptográficas, principal campo de aplicación del CRT, no tratan con datos de 32 bits, si no de 128, 256 o incluso más bits, por lo que la implementación GNU MP es la ideal para realizar las posteriores comparaciones. Además, podemos ver como no existe casi penalización en las implementaciones de este tipo de datos al aumentar el tamaño de los bits de los enteros a usar en el algoritmo, por lo que presenta un mejor rendimiento ante un aumento de las necesidades durante su ejecución.

Para realizar esta comprobación sobre el otro algoritmo diseñado e implementado, paralelización por eficiencia, se implementa una versión del mismo haciendo uso de la aritmética de enteros largos de GNU MP, y realizamos un test como el anterior para comprobar que el resultado es el esperado y podemos usar la implementación de este tipo de datos para los comparaciones entre algoritmos y frente a las implementaciones de la plataforma C++/CUDA. Los resultados pueden verse en las siguientes tablas.

Tabla 8. Módulos de 32 bits en implementación por Eficiencia.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.14	0.79
50	0.33	2.57
100	1.00	4.58
250	3.35	8.65
500	9.84	18.60
750	24.02	32.89
1000	39.67	50.40

Tabla 9. Módulos de 64 bits en implementación por Eficiencia.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.20	0.83
50	0.90	2.51
100	2.74	4.85
250	13.42	7.83
500	47.13	18.59
750	105.71	33.04
1000	182.85	53.02

Tabla 10. Módulos de 128 bits en implementación por Eficiencia.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.27	1.07
50	2.08	2.50
100	7.11	3.76
250	32.83	9.15
500	126.47	19.95
750	272.57	34.46
1000	472.81	53.22

Tabla 11. Módulos de 256 bits en implementación por Eficiencia.

Tamaño	Nativo (ms)	GNU MP (ms)
10	0.35	1.33
50	5.80	3.15
100	18.27	3.71
250	98.05	7.25
500	365.37	18.20
750	796.49	22.20
1000	1395.62	37.35

Como podemos ver se mantiene la tendencia que hemos podido observar en el algoritmo anterior, por lo que a partir de ahora todas las comparaciones se harán sobre la implementación del tipo de datos enteros largos proporcionado por la librería externa GNU MP, habiendo superado con creces el rendimiento de la implementación nativa.

C. Fuerza Bruta o Eficiencia

Ahora vamos a realizar una batería de pruebas más amplia, para decidir ante diversos casos de tamaño de enteros y número de congruencias que algoritmo presenta mejores resultados y mayor eficiencia. El estudio se realizará de forma separada en las implementaciones sobre C# y sobre CUDA.

Implementaciónes C#

Para lograr discernir que algoritmo presenta mejores resultados vamos a analizar su comportamiento ante distintas situaciones.

En primer lugar vamos a presentar los datos de las implementaciones GNU MP de ambos algoritmos recogidas anteriormente y enfrentadas para las distintas entradas. Se ha usado un caso base de 2, y el número de divisiones se ha establecido en 2 igualmente.

Tabla 12. Módulos de 32 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.92	0.66
50	2.60	4.30
100	4.72	9.80
250	8.64	32.50
500	22.62	84.52
750	30.90	148.16
1000	47.96	251.77

Tabla 13. Módulos de 64 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.93	0.66
50	2.43	4.04
100	3.38	11.98
250	8.24	30.84
500	22.42	81.22
750	31.56	155.24
1000	44.72	225.11

Tabla 14. Módulos de 128 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.76	0.66
50	3.48	4.00
100	4.52	12.20
250	8.92	30.26
500	20.16	78.80
750	33.32	155.12
1000	46.42	254.83

Tabla 15. Módulos de 256 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	1.38	0.58
50	3.24	4.08
100	5.22	9.68
250	8.56	31.72
500	24.02	98.74
750	33.50	159.28
1000	44.34	254.33

Los resultados entre ambas implementaciones distan mucho, por lo que procedemos a ajustar los valores de caso base y número de divisiones. Como vemos, en casos pequeños la diferencia no es mucha, pero al aumentar el número de datos de entrada se dispara el tiempo de ejecución por parte del segundo algoritmo. Por ello vamos a aumentar el número de divisiones de 2 a 4, y el caso base de 2 a 10. Dado que la tendencia se mantiene en todos los módulos, vamos a presentar solo los resultados para el módulo de 32 bits y ajustar los parámetros en función de ello.

Tabla 16. Módulos de 32 bits. Caso Base: 10, Divisiones: 4.

Tamaño	Eficiencia (ms) – 2,2	Eficiencia (ms) – 10,4
10	0.66	1.9
50	4.30	3.98
100	9.80	7.24
250	32.50	23.62
500	84.52	58.70
750	148.16	109.70
1000	251.77	166.06

Aumentamos aún más el caso base, ya que parece que los cambios realizados han surtido el efecto buscado, y obtenemos los siguientes resultados.

Tabla 17. Módulos de 32 bits. Caso Base: 100, Divisiones: 4.

Tamaño	Eficiencia (ms) – 10,4	Eficiencia (ms) – 100,4
10	1.9	1.48
50	3.98	3.52
100	7.24	4.46
250	23.62	18.94
500	58.70	35.30
750	109.70	74.34
1000	166.06	91.44

Esta claro que el aumento del caso base y del número de divisiones acerca los resultados a los obtenidos por la paralelización por fuerza bruta, por lo que tras algunas pruebas más se establece como caso base el factor de 1000, y como número de divisiones 6, que arroja los resultados expuestos a continuación. Se eligen estos datos básicamente porque para casos pequeños, el mejor comportamiento en estabilidad y eficiencia lo otorga el método de fuerza bruta, pero para casos de mayor tamaño, como veremos, no esta tan claro.

Tabla 18. Módulos de 32 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	1.19	0.79
50	2.74	2.57
100	4.24	4.58
250	8.99	8.65
500	20.83	18.60
750	32.79	32.89
1000	51.10	50.40

Tabla 19. Módulos de 64 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.93	0.83
50	2.43	2.51
100	4.13	4.85
250	7.77	7.83
500	19.33	18.59
750	32.13	33.04
1000	49.18	53.02

Tabla 20. Módulos de 128 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.76	1.07
50	3.15	2.50
100	2.47	3.76
250	8.85	9.15
500	19.99	19.95
750	34.59	34.46
1000	51.64	53.22

Tabla 21. Módulos de 256 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	1.23	1.33
50	2.57	3.15
100	4.17	3.71
250	8.69	7.25
500	21.31	18.20
750	34.27	22.20
1000	51.70	37.35

Una vez ajustados tanto el caso base como el número de divisiones de esta manera, podemos observar, para casos donde el módulo presenta un tamaño de hasta 128, no se aprecia mayor diferencia, pero al aumentar a 256 bits, y 500 ecuaciones en nuestro sistema de ecuaciones lineales, resulta una mejora de eficiencia de algunos ms dando como vencedor en este aspecto al método de paralelización por eficiencia, pero no siendo suficiente para llegar a inclinarse por uno u otro. Debido a ello, se proponen test más agresivos, aumentando considerablemente el número de ecuaciones del sistema hasta 20000. Los resultados se pueden ver a continuación.

Tabla 22. Módulos de 32 bits comparando Eficiencia y Fuerza Bruta, comprobando los límites del algoritmo por Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
2000	101.40	164.25
4000	253.91	405.62
6000	473.32	797.15
8000	816.34	1552.98
10000	3386.19	2518.144
15000	-	4708.26
20000	-	7836.44

Como podemos ver ya en la primera tabla de resultados, la paralelización por fuerza bruta arroja errores en proceso de ejecución debidos a los problemas comentados en el análisis teórico, propio del manejo de memoria y fallos derivados de los cálculos con un gran número de enteros largos de forma directa y, como su nombre indica, usando la fuerza bruta.

Tabla 23. Módulos de 64 bits comparando Eficiencia y Fuerza Bruta, comprobando los límites del algoritmo por Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
2000	103.00	151.00
4000	287.01	433.02
6000	545.03	858.04
8000	854.04	1572.08
10000	3254.18	2496.14
15000	-	4717.02

Se puede observar claramente cómo se mantiene la misma tendencia, donde el algoritmo por fuerza bruta arroja buenos resultados hasta una población de 10000 ecuaciones, donde cae en rendimiento frente a la implementación por eficiencia, y más allá de esa línea arroja errores de ejecución, con lo cual podemos decir que límite efectivo está por debajo de 10000 congruencias en el sistema a resolver.

Por debajo de este valor se comporta de forma adecuada e incluso supera en eficiencia al otro diseño presentado, por lo cual para esas poblaciones podría ser la elección adecuada. Aun así, conocidos los problemas del mismo con respecto a la gestión de memoria, y evaluando la diferencia, que se encuentra generalmente en menos de un segundo en las ejecuciones mencionadas, no es descartable decantarse por la implementación por eficiencia, y asegurar así la escalabilidad del sistema evitando cualquier tipo de problemática conocida.

Implementaciónes CUDA

En este caso no es necesario realizar ninguna comparativa entre tipos de datos, ya que no ha sido posible hacer uso de librerías externas para el uso de enteros largos, por lo que pasamos directamente a la comparativa entre los métodos de fuerza bruta y de eficiencia implementados sobre CUDA y C++.

Tabla 24. Módulos de 32 bits comparando Eficiencia y Fuerza Bruta.

Tamaño	F. Bruta (ms)	Eficiencia (ms)
10	0.27	1.79
50	0.34	2.48
100	0.43	3.33
250	0.48	3.25
500	0.38	4.36
750	0.51	4.09
1000	0.42	4.38
2000	0.49	5.78
4000	0.54	8.75
6000	0.69	11.09
8000	0.83	13.82
10000	0.91	15.72
15000	-	21.51
20000	-	27.77

En este caso vuelve a pasar lo mismo que en el caso de las implementaciones C# con el método de fuerza bruta, y es que a partir de una cierta cantidad conocida se vuelve inestable, arrojando resultados erróneos en este caso.

El motivo de no presentar el conjunto de tablas con los resultados para distintos tamaños de módulo es debido a que no ha sido posible contar con implementaciones externas de librerías para CUDA, debido a que CUMP, la librería wrapper que servirá de puente entre CUDA y las útiles librerías de GNU MP se encuentra en una fase de desarrollo temprana.

Por tanto, como en la implementación solo CPU, el algoritmo de Fuerza Bruta presenta grandes resultados, pero en este caso, la diferencia entre ambos algoritmos es aun menor, debido a que no obtenemos tiempos mayores de 30 ms en ningún caso, y es asumible el optar por la implementación que evitar inestabilidades con grandes conjuntos de datos. Además, como hemos podido comprobar en C#, una mejora de la implementación incluyendo GNU MP, ocasionará que, aunque aumente el tamaño de los módulos de 32 a 256 bits, el rendimiento no se verá apenas afectado en ningún momento.

Implementación C# versus CUDA

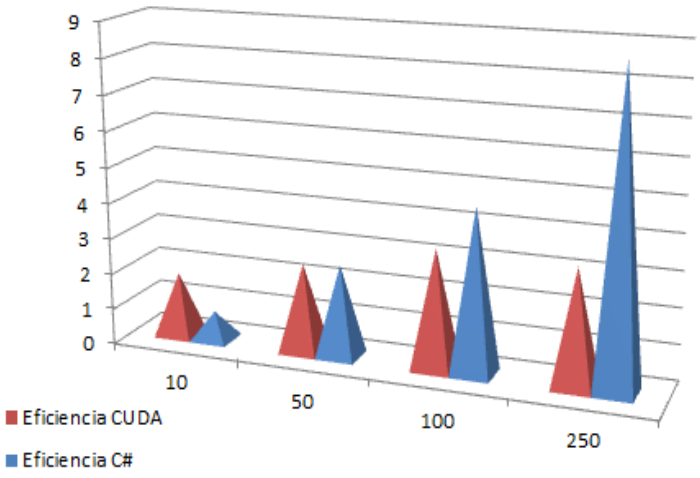
En cada una de las plataformas elegidas se ha señalado al algoritmo de paralelización por eficiencia como el más idóneo debido a su estabilidad, su eficiencia moderada, y su escalabilidad.

A continuación mostramos una tabla y una gráfica que representa las diferencias entre la implementación C# y la implementación CUDA, para el caso de módulo 32 bits, debido a las limitaciones actuales en la plataforma gráfica comentadas.

Tabla 25. Módulos de 32 bits comparando Eficiencia sobre C# y Eficiencia sobre CUDA.

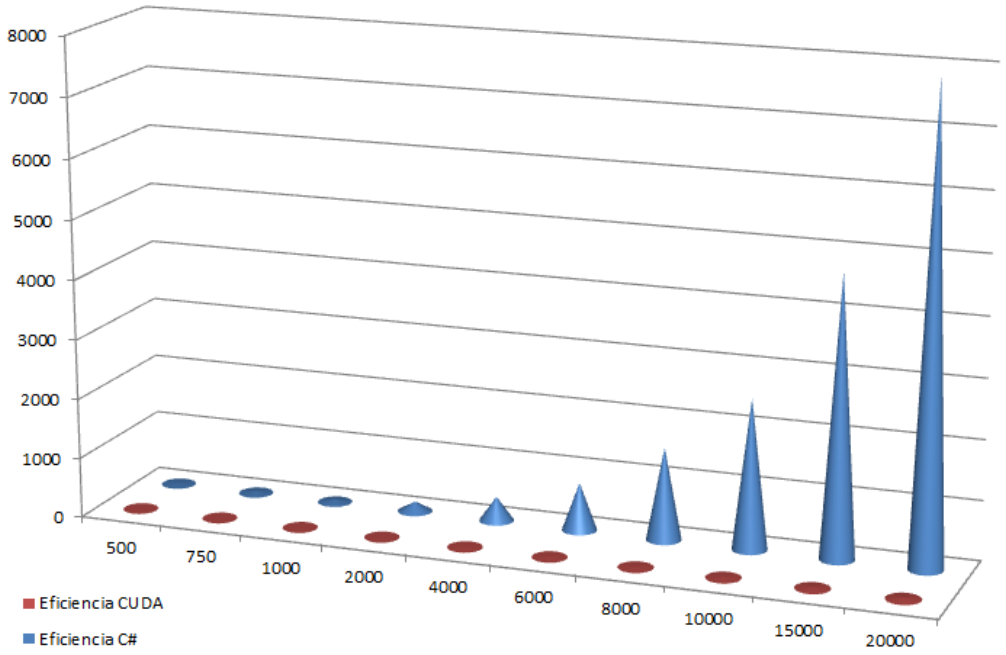
Tamaño	C# (ms)	CUDA (ms)
10	0.79	1.79
50	2.57	2.48
100	4.58	3.33
250	8.65	3.25
500	18.60	4.36
750	32.89	4.09
1000	50.40	4.38
2000	164.25	5.78
4000	405.62	8.75
6000	797.15	11.09
8000	1552.98	13.82
10000	2518.14	15.72
15000	4708.26	21.51
20000	7836.44	27.77

Figura 5. Tiempos de ejecuciones de algoritmos basados en Eficiencia sobre las plataformas C# y CUDA, desde el caso 10 al caso 250.



Sobre esta gráfica podemos observar como para casos realmente pequeños ambos se comportan de manera similar, oteniendo valores de rendimiento muy similares, pero esto empieza a cambiar a partir de 250, donde la ejecución sobre CUDA mantiene la misma tendencia y la ejecución C# se dispara en el tiempo.

Figura 6. Tiempos de ejecuciones de algoritmos basados en Eficiencia sobre las plataformas C# y CUDA, desde el caso 500 al caso 20000.



Es obvio que la implementación sobre procesador gráfico presenta unos resultados de rendimiento y eficiencia a los que no puede acceder la implementación sobre un procesador convencional.

Aun así, no debemos perder de vista la tabla y dejar que la gráfica falsee los resultados, pues en ambos casos estamos hablando de milisegundos, o pocos segundos para el caso de C# en el mayor conjunto de datos de prueba lanzado, por lo que ambas son buenas opciones y ofrecen un gran rendimiento. Aun así, actualmente la mejor opción de implementación pasa por un procesador convencional, hasta el momento en que la implementación de GNU MP este disponible para CUDA, debido a la necesidad de las aplicaciones criptográficas que hacen uso del CRT de trabajar con enteros de mayor grado de lo que la actual implementación CUDA puede proporcionar.

D. Paralelo o Secuencial

Tras exponer los resultados de toda la implementación paralela realizada, se hace necesaria alguna exposición de los resultados de una ejecución secuencial del CRT, a fin de poder establecer si el esfuerzo de paralelizar e implementar nuevas versiones del algoritmo del teorema chino de los restos ha sido suficiente para considerarlo un éxito. A tal efecto se presenta una tabla y gráfica de ejecuciones secuenciales para el CRT frente a los resultados de la implementación eficiente en C# y en CUDA.

Tabla 26. Módulos de 32 bits comparando Eficiencia sobre C#, Eficiencia sobre CUDA, y ejecución Secuencial.

Tamaño	Secuencial (ms)	C# (ms)	CUDA (ms)
10	0.10	0.79	1.79
50	0.60	2.57	2.48
100	1.70	4.58	3.33
250	8.30	8.65	3.25
500	29.00	18.60	4.36
750	66.10	32.89	4.09
1000	116.40	50.40	4.38
2000	483.82	164.25	5.78
4000	1881.10	405.62	8.75
6000	4065.43	797.15	11.09
8000	7334.21	1552.98	13.82
10000	-	2518.14	15.72
15000	-	4708.26	21.51
20000	-	7836.44	27.77

Figura 7. Tiempos de ejecuciones de algoritmos basados en Eficiencia sobre las plataformas C# y CUDA frente a ejecución secuencial, desde el caso 10 al caso 1000.

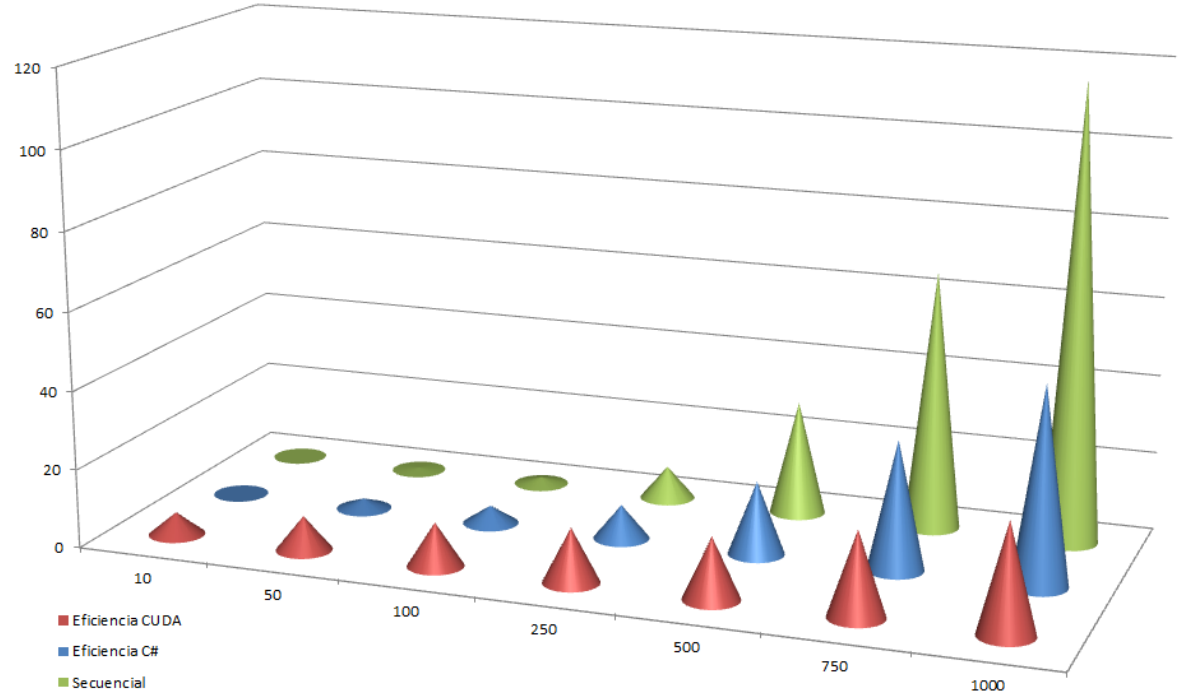
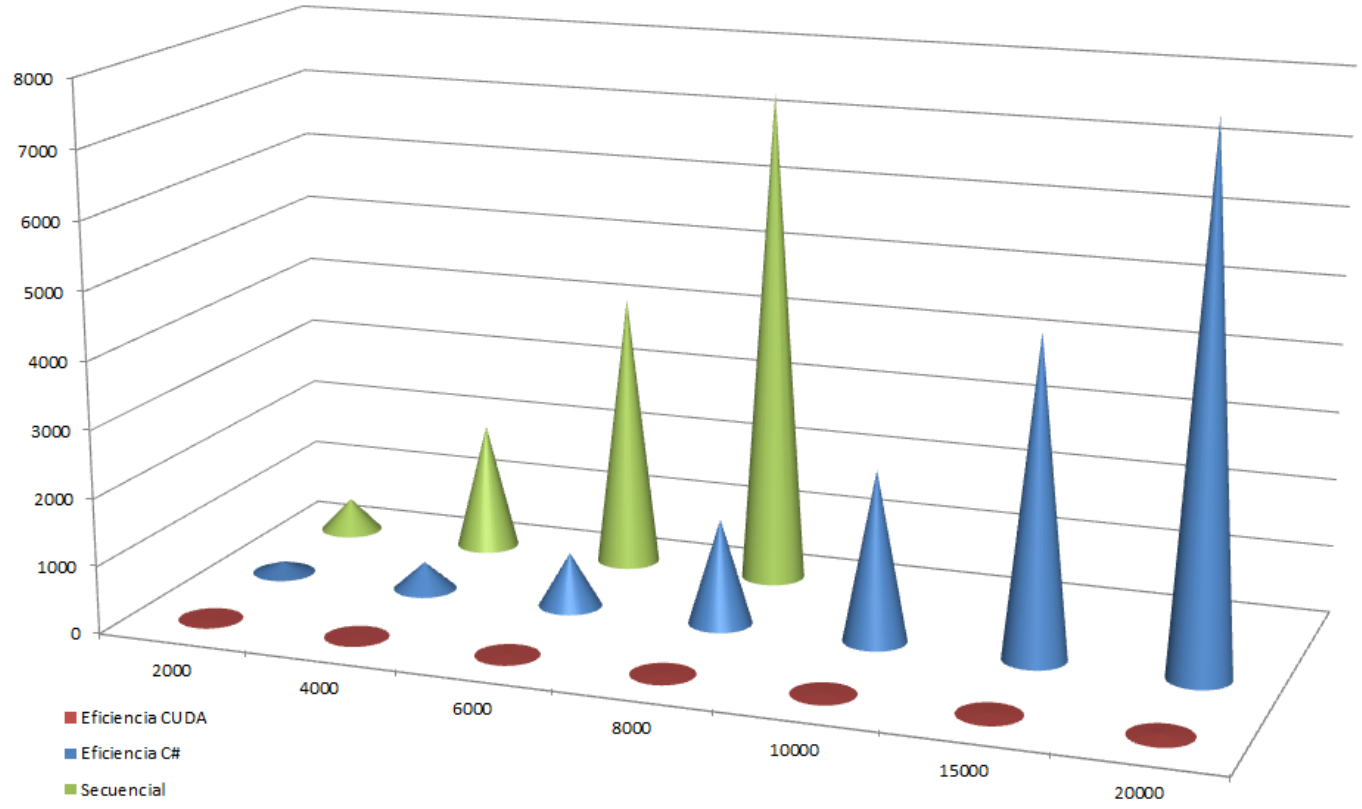


Figura 8. Tiempos de ejecuciones de algoritmos basados en Eficiencia sobre las plataformas C# y CUDA frente a ejecución secuencial, desde el caso 2000 al caso 20000.



Como se puede observar, para tamaños suficientemente pequeños, una ejecución secuencial sobre un procesador de cierta velocidad siempre será más eficiente que una creación de hilos constante que consume tiempo y recursos. Incluso a la ejecución sobre el procesador gráfico le cuesta adelante a la ejecución secuencial a bajo nivel de datos, debido a que a este nivel no se aprovechan sus capacidades, y se desaprovechan los recursos disponibles.

Cuando el conjunto crece, para el valor de 4000, se observa como ya estamos barajando tiempos de casi dos segundos para realizar el cálculo del CRT, mientras que en C# no llega al medio segundo, y en CUDA no llega a los 10 ms. Es importante señalar este hecho, pues si se usa en aplicaciones criptográficas que, como se vio en la introducción, se usen para realizar streaming de video, una espera de dos segundos para un cálculo de este tipo es demasiado tiempo. A partir de 6000 ya se empieza a disparar a los cuatro segundos y la tendencia crece resultando imposible obtener valores por encima de 8000.

Por tanto queda demostrada la eficacia de una paralelización eficiente sobre el CRT, y las presentadas cumplen con los objetivos establecidos desde un principio, estableciendo un nivel de mejora y rendimiento muy superior a los secuenciales, y además estableciendo una nueva vía, procesamiento gráfico, para conseguir niveles de eficiencia y rendimiento subiendo aún más niveles.

V. CONCLUSIONES

En base al estado actual de las implementaciones conocidas del Teorema Chino de los Restos, y a las características de estas, ya sean paralelas o secuenciales, los diseños y soluciones aportados presentan un gran avance en cuanto a términos de eficiencia y rendimiento se refiere. Además, se han establecido rangos de actuación para las distintas soluciones aportadas, en base a los cuales se puede extraer el mayor beneficio productivo de la implantación de las mismas según sus directrices.

De la misma manera, se ha logrado presentar una solución altamente escalable y adaptable a la arquitectura y necesidades del sistema en el que se acopla, superando las limitaciones de las soluciones tanto hardware como software, todas ellas muy dependientes de los recursos y los datos manejados, generando problemas altos de escalabilidad sobre todo a estos últimos.

Por tanto se considera que se han alcanzado los objetivos propuestos de forma bastante satisfactoria, aunque se trate de una aproximación con terreno aún por explorar en diversos aspectos de la eficiencia, sobre todo en lo referente a las implementaciones basadas en procesadores gráficos, cuya eficiencia mejora de forma constante conforme la plataforma que le da soporte avanza, y sus SDK se vuelven cada vez más refinados.

VI. APLICACIONES Y FUTUROS TRABAJOS

Los siguientes pasos a realizar sobre este trabajo se enmarcan en dos grandes líneas en lo que a mejora de eficiencia y rendimiento se refiere, aunque ambas están enfocadas a la plataforma GPU. En primer lugar, se hace imperativo dedicar esfuerzos a la mejora de los tipos nativos de enteros, llevando a cabo una migración de estos hacia las nuevas opciones que proporciona CUMP, como enlace entre CUDA y GNU MP, usado en las implementaciones solo CPU, dándole potencia de cálculo y escalabilidad de tamaño de datos a esta plataforma, de forma que pueda dar pie a extrapolar los buenos resultados obtenidos sobre mayores enteros. En segundo lugar, es necesario un estudio más profundo de las plataformas GPU y de las nuevas funcionalidades proporcionadas por los nuevos SDK en desarrollo, a fin de conseguir la mejor adaptabilidad sea cual sea la versión de procesador gráfico sobre la que se realice la ejecución del algoritmo implementado. De esta forma, se lograrían los mismos principios ya establecidos en las versiones de procesador convencionales.

A pesar de todo esto, algunas implementaciones son ya aplicables y, pese a las múltiples aplicaciones posibles, nos centraremos en dos relacionadas con la distribución de contenidos en esquemas multicas seguros, como se estableció en la introducción, para ejemplificarlo.

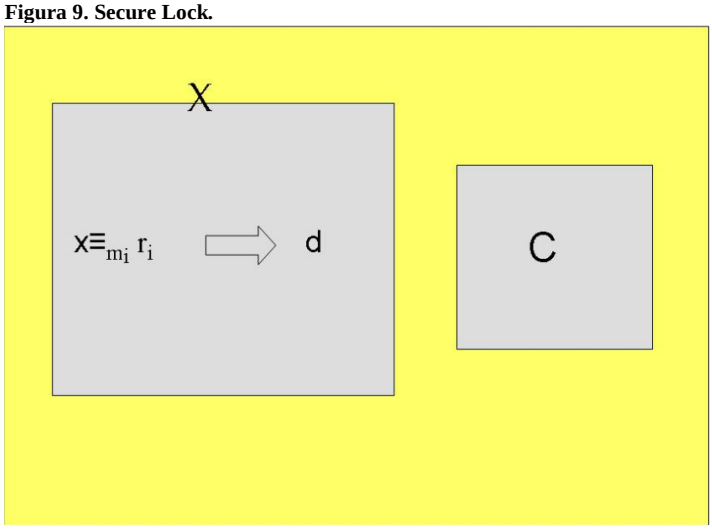
La primera hace referencia precisamente al esquema introducido en [ChiCh]. En este trabajo los autores proponen un esquema de distribución de la información seguro basado en el CRT conocido como Secure Lock. La información es cifrada utilizando una clave de sesión, obteniéndose un texto cifrado que se enviará a una pluralidad de usuarios y solamente aquellos que se encuentren en posesión de una autorización para acceder a la misma serán capaces de recuperar la información. Por tanto, se hace necesario que estos usuarios autorizados tengan un modo de acceder a la clave con la que se cifró la información distribuida. El modo para ello es proporcionar a los usuarios autorizados una información adicional que permita el acceso a la clave de sesión "encerrada" por un candado que cada usuario autorizado será capaz de abrir utilizando dicha información adicional personal. El método para construir el candado es como sigue:

- En primer lugar, la clave de sesión $\boldsymbol{K_s}$ es cifrada utilizando una clave $\boldsymbol{K_i}$, distinta para cada uno de los usuarios autorizados, de un criptosistema simétrico, por ejemplo DES ([RA]) o, tal y como recomienda la directiva actual de la NSA en cuanto a cifrado simétrico, AES ([RA]). De este modo se obtiene un valor $\boldsymbol{N_i = EK_i(K_s)}$ distinto para cada usuario.
- Utilizando un primo distinto también para cada uno de los usuarios $\boldsymbol{m_i}$ se obtiene una solución del sistema de congruencias $\boldsymbol{x \equiv N_i(mod\ m_i)}$.
- Esta solución del sistema de congruencias constituye el candado anteriormente referenciado y se distribuye junto a la información cifrada tal y como se muestra en la figura 9.

Cuando un usuario autorizado recibe el mensaje representado en la figura, utiliza su primo m_i para obtener el valor N_i y entonces, usando la clave personal K_i es capaz de descifrar N_i,

obteniendo la clave de sesión K_S utilizada para cifrar la información distribuida.

Figura 9. Secure Lock.



Sin embargo, tal y como se apunta en [KM], el sistema se vuelve ineficiente conforme el número de usuarios autorizados crece, debido principalmente al uso del CRT para la construcción del candado. De este modo, en [SLBE] los autores proponen una modificación del esquema combinado con técnicas de distribución de usuarios por grupos basados en esquemas de árbol para ofrecer un rendimiento acorde a los requerimientos de las comunicaciones actuales con un gran número de usuarios en línea y con gran dinamismo en dichos grupos en cuanto a su composición. Desafortunadamente el esquema nuevo puede verse seriamente comprometido en cuanto a su eficiencia en situaciones en las que el número de usuarios puede llegar a ser de cientos de miles o incluso millones como podría ser un importante evento deportivo, y esto debido nuevamente a la utilización del CRT en la construcción del candado. Los resultados ofrecidos en la sección IV (Tablas 25 y 26) nos llevan a concluir que podemos ofrecer servicio a un número bastante considerable, y más aún si lo combinamos con dichos esquemas de distribución de usuarios por grupos. Concretamente de las tablas expuestas en la sección anterior podemos estimar que utilizando la arquitectura de procesador convencional, dicho esquema podría renovar las claves en tan solo 4 o 5 milisegundos, mientras que si nos vamos al esquema de procesador gráfico, una vez realizadas las labores antes comentadas, podría producirse esta misma situación en tan solo 2 o 3 milisegundos. Esto supone que usando un esquema de árbol para la distribución de usuarios podríamos dar servicio a cientos de millones de usuarios de un modo totalmente eficiente. Baste como ejemplo un árbol de distribución de usuarios de tan solo profundidad 3 como los usados en [SLBE] y con un número de hijos por nodo de 1000. En este caso, la audiencia podría llegar a mil millones de usuarios.

En la segunda aplicación consideramos a un esquema de autentificación de mensajes introducido en [AL]. En dicho esquema cada usuario $u_i, i = 1, \dots, n$ tiene un primo que se usa para acceder a los mensajes de refresco de la clave de sesión de un esquema multicast tal y como se cita también en [AL], así como un segundo entero b_i en el rango $[0, x_i - 1], i = 1, \dots, n$. Entonces la clave de sesión viene dada por $r \equiv g^k mod\, m$, donde m es un primo público (puede consultarse [AL] para más detalles sobre el mismo). La información que permite la autentificación de r distribuida por el servidor de claves se genera entonces del modo siguiente:

- En primer lugar el servidor de claves genera a aleatorio tal que $a < x_i$ para todo $i = 1, \dots, n$.
- El servidor calcula $h(a)$ para h una función hash.
- Finalmente, el servidor calcula $s = (g^k)^{-1} mod\, L$, donde $L = \prod_{i=1}^{n} x_i$ y resuelve el sistema de congruencias $x \equiv as + b_i mod\, x_i, i = 1, \dots, n$, obteniendo una solución S.

La información que permite la autentificación del mensaje de refresco de la clave r es entonces el par $(S, h(a))$.

De este modo, basándonos de nuevo en los resultados ofrecidos sobre CRT en este trabajo, generaremos mensajes de autentificación de un modo eficiente y escalable.

REFERENCIAS

[AAL] J. M. Arrufat, J.A. Älvarez-Bermejo y J.A. López-Ramos Una implementación paralela del CRA con aplicaciones criptgráficas. VIII Jornadas de Matemática Discreta y Algorítmica, Almeria 2012, pp.267-274, (2012).

[AL] N.Antequera and J.A. Lopez-Ramos Remarks and countermeasures on a cryptoanalysis of a secure multicast protocol. Porceedings of 7th International Conference on Next Generation Web Services practices, Salamanca 2011, pp. 201-205, (2011).

[B] Juan de Burgos Cálculo Infinitesimal de Una Variable, Capitulo 5. McGraw-Hill (2002).

[BBBC] Jirí barnat, Petr Bauch, Lubos Brim and Milan Ceska Employing Multiple CUDA Devices to Accelerate LTL Model Checking. Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on, pp.259-266, (2010).

[Ch] H.Chen CRT-Based High-Speed Parallel Architecture por Long BCH Encoding. Circuits and Systems II: Express Briefs, IEEE Transactions on, vol.56, 9, pp.684-686, issn 1548-7747, (2009).

[ChCh] K.-C. Chan and S.-H.G. Chan. Key management approaches to offer data confidentiality for secure multicast. IEEE Network, 17(5) 30–39 (2003).

[ChiCh] G.Chiou and W.Chen Secure broadcasting using the secure lock. IEEE Trans. Softw. Eng., vol 15(8), pp. 929-934, (1989).

[ChKL] C. C. Chang, Y. T. Kuo and Y. P. Lai Parallel Computation of Residue Number System. Computing & Informatics, ICOCI '06, International Conference on, pp.1-6, (2006).

[G] Johann Grobschadl The Chinese Remainder Theorem and its Application in a High-Speed RSA Crytp Chip. Computer Security Applications, ACSAC '00, 16th Annual Conference, pp.384-393, (2000).

[KM] S.Kruus and J. P. Macker Techniques and issues in multicast security. Proceedings of Military Communications Conference, MILCOM, pp.1028-1032, (1998).

[KR] N. P. Karunadasa Accelerating High Performance Applications with CUDA and MPI. Industrial and Informations System (ICIIS), 2009 International Conference on, pp.331-336, (2009).

[LCh] Yeu-Pong Lai and Chin-Chen Chang Parallel computational algorithms for generalized Chinese remainder theorem. Computers & Electrical Engineering, vol29(8), pp.801-811, (2003).

[LK1] K.-Y. Lin and B. Krishna and H. Krishna Rings, fields, the Chinese remainder theorem and an extension-Part I: Theory. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, vol.41(10), pp. 641-655. issn 1057-7130, (1994).

[LK2] K.-Y. Lin and B. Krishna and H. Krishna Rings, fields, the Chinese remainder theorem and an extension-Part II: applications to digital signal processing. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, vol.41, 10, pp. 656-668. issn 1057-7130, (1994).

[LXWT] Y. Li, Limin Xiao, Z. Wang and H.Tian High Performance Point Multiplication for Conic Curves Cryptosystem Base on Standard NAF Algorithm and Chinese Remainder. Information Science and Applications (ICISA), 2011 International Congerence on, pp.1-8, (2011).

[LZJ] B. Liu, W. Zhang and T. Jiang A Scalable Key Distribution Scheme for Conditional Access System in Digital Pay-TV System. IEEE Consumer Electronics, vol50(2), pp.632-637, (2004).

[MA] Svelin A. Manavski CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography. Signal Processing and Communications, ICSPC 2007, pp.65-68, (2007).

[MY] Yasuyuki Murakami A New Construction Method of Knapsack PKC Using Linear Transformation and Chinese Remainder Theorem. Information Theory and its Applications (ISITA), 2010 International Symposium on, pp.433-436, (2010).

[O] Amos O. Olagunju A computational exploration of the Chinese remainder theorem. J. Appl. Math. & Informatics, vol26, pp.307-316, (2008).

[R] Michael O. Rabin Probabilistic algorithm for testing primality, Journal of Number Theory, vol12(1), pp.128-138, (1980)

[RA] Jorge Ramió Aguirre Libro Electrónico en Seguridad Informática y Criptografía, Capítulo 12. Escuela Politécnica de Madrid, (2006).

[RH] S. Rafaeli and D. Hutchison. A Survey of Key Management for Secure Group Communication. ACM Computing Surveys, 35(3) 309--329 (2003).

[RK] Kenneth H. Rosen Discrete Mathematics and Its Applications, Capítulo 4. McGraw-Hill (2012).

[SLBE] O. Scheikl, J. Lane, R. Boyer and M. Eltoweissy, Multi-level secure multicast: the rethinking of secure locks, Parallel Processing Workshops, 2002. Proceedings. International Conference on, 2002, 17-24.

[TSA] H. Toyoshima and K.Satoh and K.Ariyama High-speed hardware algorithms for Chinese remainder theorem. Circuits and Systems, ISCAS '96, Connecting the World, 1996 IEEE International Symposium on, vol2, pp.265-268, (1996).

[W] Yuke Wang New Chinese Remainder Theorems. Signals, Systems & Computers, Conference Record of the Thirty-Second Asilomar Conference on, vol1, pp.165-171, (1998).

[WWX] Wenjie Wang, Chen Wang and Xiang-Gen Xia A Robust quantization method using a robust chinese remainder theorem for secret key generation. Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, pp.1904-1907, (2011).

[ZHChFYY] Yong Zhao, Zhou Huang, Bin Chen, Yu Fang, Menglong Yan and Zhenzhen Yang Local acceleration in Distributed Goegraphic Information Processing with CUDA. Geoinformatics, 2010 18th International Conference on, pp.1-6, (2010).

[ZJ] S. Zhu and S. Jajodia. Scalable group key management for secure multicast: A taxonomy and new directions. In: Huang, H., MacCallum, D., Du, D.-Z. (eds.) Network Security, Springer, US, 57–75 (2010).

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Estudio, diseño e implementación de nuevos enfoques paralelos software del el Teorema Chino del Resto sobre diferentes plataformas y arquitecturas de tipo paralelo. Algunos de estos diseños típicos generan problemas conocidos de almacenamiento y gestión de la memoria en su ejecución al manejar enteros demasiado largos. Este problema lleva al diseño de nuevos métodos más eficientes y refinados que vienen a solventar estas cuestiones de forma original y escalable. Los resultados obtenidos no solo mejoran de forma sustancial la implementación secuencial del mismo, sino que representan un avance en eficiencia, rendimiento y escalabilidad respecto a otras alternativas existentes actualmente.

This work presents a study, design and implementation of a new parallel software approach to the Chinese Remainder Theorem on different parallel architectures. Some of the known and legacy implementation described suffer from storage and memory management issues when handling big integers, affecting substantially to performance. This problem leads to the design of a new and more efficient and refined method to address these issues in a scalable way. The results not only substantially overcome the sequential implementation but represent an doubtlessly enhancement in efficiency, performance and scalability regarding existing alternatives